VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

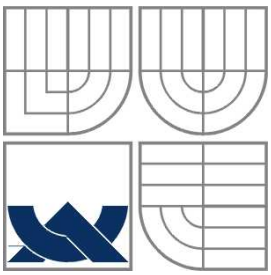
MODUL SHLUKOVÉ ANALÝZY SYSTÉMU PRO DOLOVÁNÍ Z DAT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

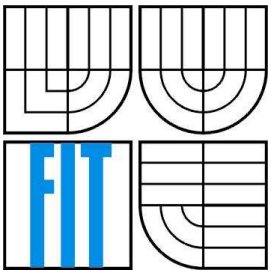
AUTOR PRÁCE
AUTHOR

Bc. PAVEL RIEDL

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MODUL SHLUKOVÉ ANALÝZY SYSTÉMU PRO DOLOVÁNÍ Z DAT

CLUSTER ANALYSIS MODULE OF A DATA MINING SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. PAVEL RIEDL
AUTHOR

VEDOUCÍ PRÁCE — Doc. Ing. JAROSLAV ZENDULKA, CSc.
SUPERVISOR

BRNO 2010

Abstrakt

Tato diplomová práce se zabývá vývojem modulu pro systém dolování z dat, jenž je vyvíjen na FIT. První část se věnuje obecnému procesu získávání znalostí a shlukové analýze včetně validace shluků, popisuje také Oracle Data Mining včetně algoritmů, které používá pro shlukování. Na závěr představuje vyvíjený systém a jím používané technologie, kterými jsou platforma NetBeans a jazyk DMSL. Druhá část popisuje návrh modulu shlukové analýzy a modulu pro porovnání jejích výsledků. Věnuje se také vizualizaci výsledku shlukové analýzy a ukazuje dosažené cíle.

Abstract

This master's thesis deals with development of a module for a data mining system, which is being developed on FIT. The first part describes the general knowledge discovery process and cluster analysis including cluster validation; it also describes Oracle Data Mining including algorithms, which it uses for clustering. At the end it deals with the system and the technologies it uses, such as NetBeans Platform and DMSL. The second part describes design of a clustering module and a module used to compare its results. It also deals with visualization of cluster analysis results and shows the achievements.

Klíčová slova

dolování dat, získávání znalostí z databází, modul, shlukování, shluková analýza, míry podobnosti, validace shluků, NetBeans, Oracle Data Mining, validační indexy

Keywords

data mining, knowledge discovery in databases, module, clustering, cluster analysis, proximity measures, cluster validation, NetBeans, Oracle Data Mining, validation indices

Citace

Riedl Pavel: Modul shlukové analýzy systému pro dolování z dat, diplomová práce, Brno, FIT VUT v Brně, 2010

Modul shlukové analýzy systému pro dolování z dat

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Doc. Ing. Jaroslava Zendulky, CSc.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Pavel Riedl
26. května 2010

Poděkování

Zde bych rád poděkoval panu Doc. Ing. Jaroslavu Zendulkovi, CSc za poskytnutou pomoc.

Obsah

1 Úvod

Koncem 20. století jsme se stali svědky obrovského nárůstu objemu uchovávaných dat. Data jsou zaznamenávána i v situacích, ve kterých si to běžně neuvědomujeme, ať už jsou to nákupy v obchodech, nebo třeba běžné telefonáty. Dalo by se říci, že se v těchto datech doslova topíme a tak se zde objevila otázka, zda by nebylo možné tato data nějakým způsobem využít a vyčíst z nich informace, které by pro nás mohly znamenat nějakou výhodu. To bylo motivací pro vznik nového oboru – dolování dat.

Dolování se však potýká s řadou nepříjemností způsobenou tím, že data mohou být uložena v „nehezké“ podobě. Tím jsou zde myšleny problémy s konzistencí, nevhodným formátem, apod. Tyto problémy řeší předzpracování dat, které je spolu s dolováním součástí komplexního procesu označovaného jako získávání znalostí z databází (Knowledge discovery in databases – KDD). Tento proces představuje ucelené navazující kroky k získání znalosti, kterou můžeme následně využít ve svůj prospěch. Pojem dolování dat bývá mnohdy chápán jako synonymum pro celý tento proces.

Získávání znalostí z databází již našlo své využití v celé řadě oborů a odvětví. Dnes se často využívá při analýze kriminality a detekci podvodů, k analýze nákupního košíku, ve finanční sféře, v medicíně, v bioinformatice a dalších. Nejen v posledním ze jmenovaných figuruje i shluková analýza, jejímž úkolem je nalézt v datech skupiny objektů, které se co nejvíce podobají.

Dolování se často provádí s pomocí specializovaných nástrojů, které umožňují provedení více dolovacích úloh. Jedním z těchto nástrojů je i systém vyvíjený na Fakultě informačních technologií Vysokého učení technického v Brně. Tento systém je postaven na platformě NetBeans a pro ukládání a práci s daty využívá podpory databázového systému firmy Oracle. V současnosti však tento systém není zcela dokončen, umí však již provádět úlohy týkající se dolování asociačních pravidel, klasifikace a predikce. Jsou tedy dále prováděny změny v souvislosti s úpravami jádra a vývojem nových dolovacích modulů.

Procesem dolování dat se bude zabývat kapitola 2. Shlukovou analýzou a validací získaných shluků kapitola 3. Protože modul vyvíjený v rámci této práce bude využívat algoritmy implementované v Oracle Data Mining (ODM), bude ODM včetně těchto algoritmů popsáno v kapitole 4. Kapitola 5 se bude věnovat popisu systému vyvíjeného na FIT. Kapitola 6 se bude věnovat návrhu modulu shlukové analýzy. Prezentaci výsledů uživateli a vizualizaci bude popisovat kapitola 7. V kapitole 8 popíšeme návrh modulu pro porovnání výsledků a v 9. kapitole si na příkladě ukážeme jak používat nové moduly, které vznikly v rámci této práce.

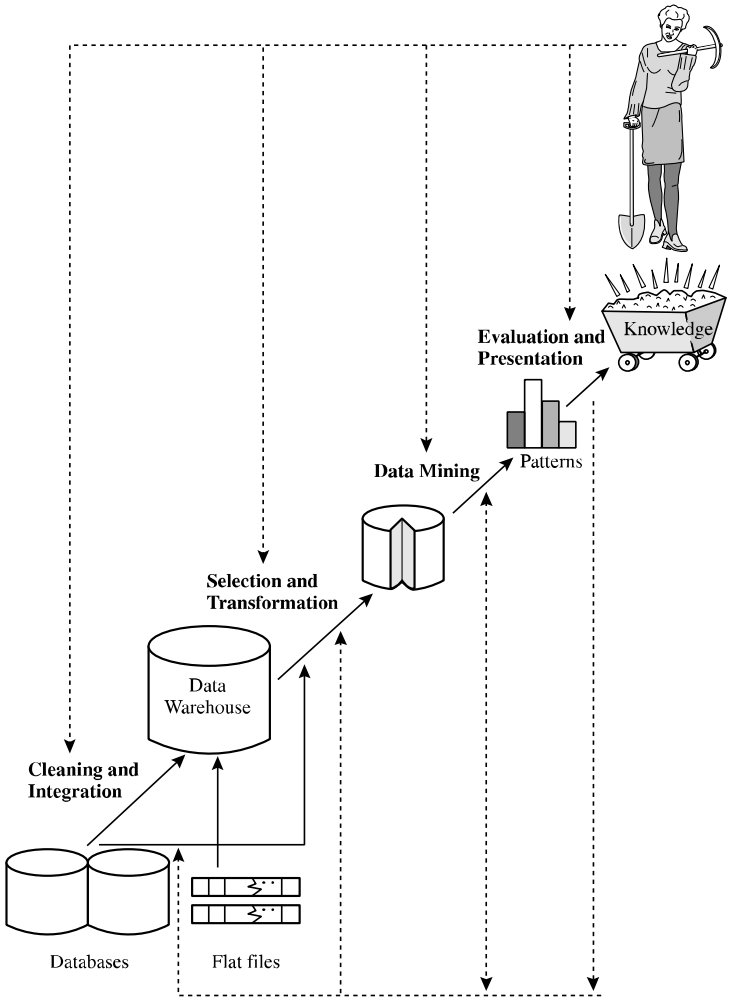
2 Získávání znalostí z databází

Můžeme říci, že získávání znalostí z databází je extrakce (neboli „dolování“) zajímavých (netriviálních, skrytých, dříve neznámých a potenciálně užitečných) modelů dat a vzorů z velkých objemů dat [1]. Takto extrahované vzory, které splňují předešlé podmínky, nám pak poskytují znalosti.

Poměrně důležitý je zde i fakt, že tyto znalosti nelze získat obyčejným SQL dotazem nad databází. Vysoký význam přísluší i užitečnosti takto získaných informací, které mohou mít vliv na důležitá rozhodnutí.

2.1 Dolování z dat jako proces

Pojem *dolování z dat* je často chápán jako synonymum pro celý proces získávání znalostí. Jindy je ale brán pouze jako jeden krok v tomto procesu. Tento proces zobrazuje obrázek 2.1.



Obrázek 2.1: Dolování dat jako proces (převzato z [2])

Jak lze z obrázku vidět, získávání znalostí z dat je sekvencí několika na sebe navazujících kroků. Mezi tyto kroky patří:

1. **Čištění dat** (data cleaning) značí odstranění chybějících či odlehlých hodnot a nekonzistence.
2. **Integrace dat** (data integration) sjednocuje data z různých zdrojů.
3. **Výběr dat** (data selection) označuje volbu pouze těch dat, která jsou pro danou dolovací úlohu relevantní.
4. **Transformace dat** (data transformation) převádí data na tvar vhodný k řešení dané úlohy.
5. **Dolování (z) dat** (data mining) je hlavní proces, ve kterém dochází k extrakci vzorů.
6. **Hodnocení vzorů** (pattern evaluation) identifikuje vzory, které jsou opravdu zajímavé a reprezentují tak znalost.
7. **Prezentace znalostí** (knowledge presentation) je také velice důležitá. Vhodným způsobem prezentuje získané znalosti uživateli.

Kroky 1 až 4 bývají také označovány jako předzpracování dat.

2.2 Předzpracování dat

Předzpracování je nedílnou součástí získávání znalostí z dat. Data, která máme k dispozici, nemusejí být (a také většinou nejsou) k dolování vhodná. A tak je potřeba, abychom je nejprve pro samotné dolování připravili. Data z reálného světa trpí celou řadou problémů. Často bývají zašuměná, některé jejich hodnoty chybí, nebo jsou nekonzistentní v důsledku toho, že byla získána z více zdrojů. Výše uvedené problémy řeší **čištění dat**. Samotné spojení dat získaných z více zdrojů se nazývá **integrace dat**. Další částí předzpracování je **transformace dat**. Ta se stará právě o to, aby měla data vhodnou podobu pro danou dolovací úlohu. Mezi metody transformace patří zejména *normalizace* a *agregace*. V poslední řadě je potřeba ze vzorku dat eliminovat nadbytečné hodnoty či atributy, tento proces označujeme jako **výběr (redukci) dat**. Těmito postupy se blíže zabývají následující kapitoly.

2.2.1 Čištění dat

Postupy čištění dat se snaží zejména o doplnění chybějících hodnot, vyhlazení šumu v datech, identifikaci odlehlých hodnot a odstranění nekonzistence.

2.2.1.1 Odstranění chybějících hodnot

Pokud u některého z atributů chybí veliký počet hodnot, může to mít negativní vliv na konečnou kvalitu výsledku. Proto vznikla celá řada způsobů, kterými se snažíme s chybějícími hodnotami vypořádat. Můžeme například:

- **Ignorovat celou n-tici**. Tato metoda není příliš efektivní. Použitelná je pouze, pokud v dané n-tici chybí hodnoty pro příliš mnoho atributů. Tato metoda by také mohla vést na příliš velikou redukci dat.
- **Doplnit chybějící hodnoty manuálně**, což je v praxi nepoužitelný přístup, kvůli vysoké časové náročnosti.
- **Vyplnit chybějící hodnoty konstantou**. Pokud by se nová konstanta vyskytovala v datech příliš často, mohl by ji dolovací algoritmus považovat za významnou. Tímto způsobem by byl výsledek negativně ovlivněn.
- **Použít průměrnou hodnotu**. V tomto případě jsou chybějící hodnoty nahrazeny aritmetickým průměrem nechybějících hodnot.
- **Použít průměrnou hodnotu všech vzorků, náležících do stejné třídy jako daná n-tice.** Tuto metodu je nejlepší ilustrovat na příkladě. Pokud bude n-tice obsahovat atributy *plat* a *povolání*, hodnota povolání bude *horník* a plat bude chybět. Tak plat nahradíme průměrem platu horníků, nikoliv celé populace.
- **Použít nejpravděpodobnější hodnotu.** Tento způsob nahrazování je již složitější. Chybějící hodnota může být určena například pomocí metod klasifikace a predikce.

2.2.1.2 Odstranění šumu

Šum může znamenat chybu, nebo vysokou odchylku v hodnotě atributu. Příliš odlehlé hodnoty mohou ovlivnit kvalitu výsledku dolování. Opět zde existuje více metod, kterými se můžeme se šumem vyrovnat:

- **Plnění (binning)** – Tato metoda vyhlazuje hodnoty zohledněním jejich okolí. Seřazené hodnoty jsou nejprve rozděleny do několika košů tak, že v každém koši je přibližně stejný počet hodnot. Poté se hodnota každého prvku nahradí buď průměrem celého koše, nebo hraniční hodnotou, ke které je nejblíže. Tím, že metody plnění nahlížejí do svého okolí hodnot, provádějí *lokální vyhlazení* [2].
- **Regrese** – Aproximuje hodnoty pomocí křivky. V případě, že se jedná o lineární regresi, jsou hodnoty proloženy přímkou.
- **Shlukování** – V tomto případě se používá k detekci odlehlých objektů, které mohou být z datového vzorku později vynechány, nebo nahrazeny. K tomuto účelu je vhodné použít shlukovací metody založené na hustotě. Shlukování se obecně dále věnuje kapitola 3.

2.2.2 Transformace a integrace

Při dolování z dat často potřebujeme *integrovat* data z více zdrojů. Také je možné, že budeme tato data potřebovat *transformovat* do podoby vhodné k dolování.

2.2.2.1 Integrace

Prvním problémem, se kterým se můžeme u integrace setkat, jsou *konflikty schématu*. Jako příklad pro tento konflikt nám poslouží adresa, která může být v první tabulce vedena jako samostatný sloupec, ve druhé může být rozepsána na ulici, město, atd. U *konfliktů hodnot* se jedná o různý formát hodnot (formát data, roku, …) ve sloupci se stejným významem. Docházet může dále i ke *konfliktům identifikace* (např. id x rodné číslo) a k *redundanci.*

2.2.2.2 Transformace

Při *transformaci dat* převádíme data do podoby vhodné k dolování. Mezi transformace dat patří:

- **Vyhlazování** či také odstranění šumu, tak jak bylo popsáno v kapitole 2.2.1.2.
- **Agregace** slučuje více hodnot dohromady, výslednou hodnotu určí pomocí nějaké operace (funkce), u spojitých hodnot se může jednat třeba o sčítání. (např. denní výdělky mohou být agregovány na měsíční, roční, …)
- **Generalizace** data na nízké detailní úrovni mohou být nahrazeny vyšším konceptem. (ulice za města, kraje, …)
- **Normalizace** přepočítává data z výchozího do cílového intervalu. Typicky <-1, 1> nebo <0, 1>.
- **Konstrukce atributů** vytváří nové atributy. Rodné číslo může být například rozděleno na věk a pohlaví.

2.2.3 Redukce

Metody redukce využíváme ke snížení množství dat. Datům můžeme odebrat nadbytečné atributy, nebo snížit počet hodnot. Využívají se následující techniky:

- **Agregace datové kostky –** Využívá se v datových skladech. Operace agregace je použita k redukci dat uložených v podobě datové kostky.
- **Výběr podmnožiny atributů –** Metoda redukce, která spočívá v odstranění irelevantních, či málo relevantních atributů.
- **Redukce dimensionality –** Využívá mechanismy kódování k redukci dat.
- **Redukce počtu hodnot –** Hodnoty jsou buď nahrazeny a později odhadnuty parametrickým modelem, nebo mohou být redukovány pomocí vzorkování či shlukování.
- **Diskretizace a generování konceptuální hierarchie –** Hodnoty mohou být nahrazeny konečným počtem intervalů, nebo údajem z vyšší úrovně konceptuální hierarchie.

2.3 Typy dolovacích úloh

2.3.1 Dolování asociačních pravidel

Spočívá ve vyhledávání frekventovaných vzorů. Tím zde rozumíme množiny hodnot, které se v datech vyskytují relativně často. Typickou úlohou této kategorie je analýza nákupního košíku, která se snaží odhalit, které produkty se spolu nejčastěji objevují v jednom nákupu.

2.3.2 Klasifikace a predikce

Obě tyto techniky se snaží předpovídat hodnotu některého z atributů. Atribut, který bude předpovídán, je označen jako „cíl“ klasifikace (predikce). Rozdíl mezi těmito metodami je v tom, že klasifikace zařazuje objekt do některé z předem definovaných tříd, kdežto predikce určuje spojitou hodnotu.

Data pro klasifikaci a predikci bývají zpravidla rozdělena do 3 množin. Na trénovací množině se algoritmus naučí klasifikovat (predikovat). Pomocí testovací množiny se ověří jeho úspěšnost. Na poslední množinu již aplikuje své naučené schopnosti.

Klasifikace a predikce často využívají metod založených na rozhodovacím stromu, neuronových sítích a regresi.

2.3.3 Shluková analýza

Tak jako klasifikace zařazovala objekty do předem známých tříd, tak shluková analýza žádné třídy předem nezná. Shlukování se tedy snaží nalézt množiny objektů, které se sobě podobají. Existuje celá řada metod, kde každá z nich chápe shluky trochu jinak. Pro každou z nich je však důležité, aby podobnost objektů uvnitř shluku byla co největší a podobnost objektů mezi shluky naopak co nejmenší. Dopodrobna se budeme shlukováním zabývat v kapitole 3.

2.3.4 Analýza odlehlých objektů

Využívá shlukování. Důležité však je, že nehledáme podobné objekty, ale takové, které se od ostatních velice liší. Své uplatnění nachází například při detekci podvodů.

2.3.5 Evoluční analýza

Využívá se při dolování dat, která jsou proměnná v čase. Sleduje, jak často a jak rychle se tyto hodnoty mění.

3 Shluková analýza

Co se skrývá za shlukovou analýzou, jsme již nastínili v předchozí kapitole. Cílem shlukování je rozdělit konečnou nepopsanou množinu dat do konečné a diskrétní množiny „přirozených“, schovaných datových struktur, raději než poskytovat přesnou charakterizaci nerozpoznaných vzorů generovaných stejnou distribucí pravděpodobnosti [3]. Předchozí definice srovnává shlukování s klasifikací. Tím, že je množina nepopsaná, se myslí právě to, že zde nejsou dané žádné třídy, do kterých bychom objekty rozřazovali. Dále také nastiňuje, že klasifikace může na základě pravděpodobnosti chybně zařadit objekt do třídy, do které nepatří. Nicméně u klasifikace můžeme pomocí testovací množiny jednoduše určit, jak kvalitně je schopna jednotlivé objekty rozřazovat. U shlukové analýzy se používají zcela odlišné postupy. Hodnocení kvality shluků se bude věnovat kapitola 3.3.

Víme tedy, o co se shluková analýza snaží. Nyní je na čase, povědět si, co přesně tedy je shluk. Shluk je množina entit, které jsou si podobné a entity z jiných shluků si nejsou podobné [3]. Jakým způsobem ale určíme, které objekty jsou si podobné? Pomocí tzv. měr podobnosti (proximity measures), mezi které patří vzdálenostní a podobnostní funkce. Vzdálenostní se používají především u numerických typů proměnných, podobnostní u kategorických. Čím vyšší hodnotu vzdálenosti dostaneme, tím se objekty méně podobají. U podobnosti je tomu ale naopak, proto se často využívá tzv. odlišnost. Tu není složité vypočítat, lze ji získat odečtením podobnosti od čísla 1:

$$d(\mathbf{x}_i, \mathbf{x}_j) = 1 - S(\mathbf{x}_i, \mathbf{x}_j) \tag{3.1}$$

Přesto že tato odlišnost nabývá hodnot od 0 do 1, budeme ji dále označovat jako vzdálenost. Měrám podobnosti se budou věnovat kapitoly popisující jednotlivé typy dat při shlukování.

3.1 Typy dat při shlukování

Při výpočtu vzdálenosti objektů je potřeba rozlišovat o jaký typ proměnných se jedná. Většinou rozlišujeme proměnné *nominální, ordinální, intervalové a poměrové*. *Binární* proměnné můžeme chápat jako druh nominálních proměnných, který nabývá pouze 2 hodnot. U každého typu se vzdálenost (podobnost) vypočítá odlišným způsobem, jsou zde však základní podmínky, které musí vzdálenostní (podobnostní) funkce splňovat. Nejprve si uvedeme podmínky pro vzdálenost. Pro všechna $\mathbf{x}_i$, $\mathbf{x}_j$, kde $\mathbf{x}_k = (x_1, x_2, x_3, \dots, x_d)$ platí:

Symetrie:

$$D(\mathbf{x}_i, \mathbf{x}_j) = D(\mathbf{x}_j, \mathbf{x}_i) \tag{3.2}$$

Nezápornost:

$$D(\mathbf{x}_i, \mathbf{x}_j) \geq 0 \tag{3.3}$$

Mohou však být uspokojeny i další podmínky, mezi které patří:

Trojúhelníková nerovnost:

$$D(\mathbf{x}_i, \mathbf{x}_j) \leq D(\mathbf{x}_i, \mathbf{x}_k) + D(\mathbf{x}_k, \mathbf{x}_j) \tag{3.4}$$

Totožnost:

$$D(\mathbf{x}_i, \mathbf{x}_j) = 0 \iff \mathbf{x}_i = \mathbf{x}_j \tag{3.5}$$

Pokud jsou uspokojeny i poslední dvě podmínky, hovoříme o vzdálenostní funkci jako o metrice. Podmínky pro podobnost se příliš neliší. Pro všechna $\mathbf{x}_i$, $\mathbf{x}_j$ platí:

Symetrie:

$$S(\mathbf{x}_i, \mathbf{x}_j) = S(\mathbf{x}_j, \mathbf{x}_i) \tag{3.6}$$

Nezápornost:

$$0 \leq S(\mathbf{x}_i, \mathbf{x}_j) \leq 1 \tag{3.7}$$

Z posledního vzorce lze vidět, že nejvyšší hodnota podobnosti je 1. Tato hodnota říká, že si jsou objekty maximálně podobné. Stejně tak, jako u vzdálenosti, lze stanovit metriku i u podobnosti. O metriku podobnosti se jedná v případě splnění následujících 2 podmínek:

$$S(\mathbf{x}_i, \mathbf{x}_j) S(\mathbf{x}_j, \mathbf{x}_k) \leq \left(S(\mathbf{x}_i, \mathbf{x}_j) + S(\mathbf{x}_j, \mathbf{x}_k)\right) S(\mathbf{x}_i, \mathbf{x}_k) \tag{3.8}$$

a

$$S(\mathbf{x}_i, \mathbf{x}_j) = 1 \iff \mathbf{x}_i = \mathbf{x}_j. \tag{3.9}$$

Shlukovací algoritmy pak operují s některou z následujících struktur:

Datová matice slouží k reprezentaci datové množiny, její rozměry jsou $n \times d$, kde *n* značí počet objektů a *d* počet proměnných (atributů, dimenzí):

$$\begin{bmatrix} x_{11} & \cdots & x_{1l} & \cdots & x_{1d} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ x_{i1} & \cdots & x_{il} & \cdots & x_{id} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nl} & \cdots & x_{nd} \end{bmatrix} \tag{3.10}$$

Vzdálenostní matice uchovává vzdálenosti mezi jednotlivými objekty. Rozměry matice jsou $n \times n$, kde *n* značí počet všech objektů, *i*-tý řádek *j-tý* sloupec pak udává vzdálenost objektů $\mathbf{x}_i$ a $\mathbf{x}_j$:

$$\begin{bmatrix} 0 & & & & \\ d(\mathbf{x}_2, \mathbf{x}_1) & 0 & & & \\ d(\mathbf{x}_3, \mathbf{x}_1) & d(\mathbf{x}_3, \mathbf{x}_2) & 0 & & \\ \vdots & \vdots & \vdots & \ddots & \\ d(\mathbf{x}_n, \mathbf{x}_1) & d(\mathbf{x}_n, \mathbf{x}_2) & \cdots & \cdots & 0 \end{bmatrix} \tag{3.11}$$

V dalších kapitolách se budeme věnovat měrám podobnosti. U numerických hodnot si budeme uvádět vzdálenost, u kategorických podobnost, přičemž budeme mít na paměti, že vzdálenost lze z podobnosti vždy získat podle vztahu 3.1. Prvním typem proměnných jsou proměnné intervalové, které se velice podobají proměnným poměrovým. Ne všechna literatura uvádí definici poměrových proměnných stejně.

Někdy se uvádí, že intervalové proměnné nemají absolutní nulu, následkem toho nemá smysl počítat poměr mezi dvěma hodnotami, má však smysl počítat rozdíl. Příkladem tohoto typu proměnné může být teplota udávaná ve stupních celsia. O teplotě 20°C se může zdát, že je 2x větší než 10°C. Pokud bychom ovšem místo stupňů Celsia používali Kelviny, nebyl by už poměr těchto teplot 2, ale 1,03. U poměrových proměnných je nula absolutní. Z toho vyplývá, že dávat 2 hodnoty do poměru smysl má. Příkladem může být třeba hmotnost, kde můžeme jednoznačně říci, že 100kg je 2x více, než 50kg. Takto jsou intervalové a poměrové proměnné definovány v [3].

Jiná literatura však uvádí, že rozdíl je v tom, že poměrové proměnné jsou uváděny v jiném, než v lineárním (nejčastěji exponenciálním) měřítku (viz [1] a [2]). Takto budeme chápat tyto druhy proměnných i v tomto textu.

3.1.1 Intervalové proměnné

Jsou takové proměnné, jež obvykle nabývají velikého množství číselných hodnot v lineárním měřítku. Vzdálenost objektů $\mathbf{x}_i$ a $\mathbf{x}_j$ s proměnnými intervalového typu se vypočítá dle následujících měr:

Nejčastější měrou u intervalových proměnných je Euklidovská vzdálenost:

$$D(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{l=1}^{d} |x_{il} - x_{jl}|^2} \tag{3.12}$$

Dále je také možno použít Manhattanskou (city block) vzdálenost:

$$D(\mathbf{x}_i, \mathbf{x}_j) = \sum_{l=1}^{d} |x_{il} - x_{jl}| \tag{3.13}$$

Obě tyto vzdálenosti lze převést na obecný tvar označovaný jako Minkowského vzdálenost:

$$D(\mathbf{x}_i, \mathbf{x}_j) = \left(\sum_{l=1}^{d} |x_{il} - x_{jl}|^p \right)^{\frac{1}{p}} \tag{3.14}$$

Kde *d* značí počet proměnných (dimenzí) objektu $\mathbf{x}_k$. Dosazením čísla 1 za parametr *p* ve vztahu pro Minkowského vzdálenost dostaneme Manhattanskou vzdálenost. To samé platí pro Euklidovskou vzdálenost po dosazení čísla 2.

3.1.2 Binární proměnné

Jedná se o proměnné, které mohou nabývat 2 hodnot. Binární proměnné se dále dělí na *symetrické* a *asymetrické*. U symetrických binárních proměnných jsou obě hodnoty stejně důležité. U asymetrických proměnných považujeme jednu hodnotu za významnější, a to zpravidla pokud tato hodnota značí výskyt nějaké vlastnosti, která je vzácná.

K výpočtu podobnosti binárních proměnných využíváme součty, které značí počet proměnných, ve kterých se objekty shodují, nebo počet proměnných, ve kterých se liší. Tyto součty ukazuje následující tabulka (Tabulka 1).

		$\mathbf{x}_i$		
		1	0	Celkem
$\mathbf{x}_j$	1	n_{11}	n_{10}	$n_{11} + n_{10}$
	0	n_{01}	n_{00}	$n_{01} + n_{00}$
	celkem	$n_{11}+n_{01}$	$n_{10}+n_{00}$	d

Tabulka 1: Kontingenční tabulka pro binární proměnné

Součet n_{11} (resp. n_{00}) značí kolikrát se na stejných místech v $\mathbf{x}_j$ *a* $\mathbf{x}_i$ vyskytuje hodnota 1 (resp. 0). Číslo n_{01} značí počet atributů, ve kterých se vyskytuje 0 u $\mathbf{x}_j$ a 1 u $\mathbf{x}_i$ (u n_{10}je to naopak). Součet všech těchto hodnot pak musí být *d*, což je počet proměnných objektů $\mathbf{x}_j$ a $\mathbf{x}_i$.

Podobnost symetrických binárních proměnných se pak vypočítá dle vztahu:

$$S(\mathbf{x}_i, \mathbf{x}_j) = \frac{n_{11} + n_{00}}{n_{11} + n_{00} + n_{10} + n_{01}} = \frac{n_{11} + n_{00}}{d} \tag{3.15}$$

Pro podobnost asymetrických binárních proměnných platí odlišný vztah (Jaccardův koeficient):

$$S(\mathbf{x}_i, \mathbf{x}_j) = \frac{n_{11}}{n_{11} + n_{10} + n_{01}} \tag{3.16}$$

3.1.3 Nominální proměnné

Nabývají konečného a nízkého počtu diskrétních hodnot, nelze nad nimi vytvořit uspořádání. Může se jednat například o výčet barev. Pokud je počet proměnných *d*. Pak se podobnost objektů $\mathbf{x}_i$ a $\mathbf{x}_j$ vypočítá pomocí vztahu:

$$S(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{d}\sum_{l=1}^{d} S_{ijl} \text{ , kde } S_{ijl} = \begin{cases} 0 \; pro \; x_{il} \neq x_{jl} \\ 1 \; pro \; x_{il} = x_{jl} \end{cases} \tag{3.17}$$

Za S_{ijl} tedy dosadíme 1, pokud se objekty v proměnné na *l*-té pozici shodují, 0 pokud se neshodují.

3.1.4 Ordinální proměnné

Od nominálních proměnných se liší v tom, že nad nimi lze vytvořit uspořádání. Příkladem může být následující výčet hodnot: *malý, střední, veliký*. Tento výčet pak můžeme nahradit hodnotami 1 ... M_l. Pro objekt $\mathbf{x}_i$ pak hodnotu proměnné na pozici *l* označujeme jako r_{il}. Tyto hodnoty pak přepočítáme do intervalu <0, 1>:

$$z_{il} = \frac{r_{il} - 1}{M_l - 1} \tag{3.18}$$

Vzdálenost pak vypočítáme pomocí libovolné vzdálenostní funkce (např. pomocí Euklidovské vzdálenosti).

3.1.5 Poměrové proměnné

Jsou proměnné, které nejsou uváděny v lineární stupnici. Možností, jak s nimi zacházet je více:

- Můžeme je zpracovat stejně jako intervalové proměnné (většinou nevhodné).
- Můžeme je zpracovat jako ordinální data.
- Můžeme na ně aplikovat nějakou transformaci a dále s nimi pracovat jako s intervalovými proměnnými. V případě, že jsou proměnné v exponenciálním měřítku, použijeme logaritmickou transformaci:

$$y_{il} = \log x_{il} \tag{3.19}$$

3.1.6 Proměnné různého typu

V praxi se nejčastěji setkáváme s takovými množinami dat, ve kterých se vyskytují proměnné různých typů. Pro podobnost takového druhu platí následující vztah:

$$S(\mathbf{x}_i, \mathbf{x}_j) = \frac{\sum_{l=1}^{d} \delta_{ijl} S_{ijl}}{\sum_{l=1}^{d} \delta_{ijl}}, \qquad kde\ \delta_{ijl} = \begin{cases} 0, pokud\ hodnota\ x_{il}\ nebo\ x_{jl}\ chybí \\ 1, jinak \end{cases} \tag{3.20}$$

Výpočet podobnosti se u tohoto vztahu liší dle typu proměnné. Pro kategorické proměnné (včetně binárních) platí:

$$S_{ijl} = \begin{cases} 1\ pro\ x_{il} = x_{jl} \\ 0\ pro\ x_{il} \neq x_{jl} \end{cases} \tag{3.21}$$

Pro numerické proměnné platí:

$$S_{ijl} = 1 - \frac{|x_{il} - x_{jl}|}{\max_n x_{nl} - \min_n x_{nl}} \tag{3.22}$$

Kde část ve jmenovateli značí celkový rozsah *l*-té proměnné.

3.2 Dělení shlukovacích metod

3.2.1 Metody založené na rozdělování

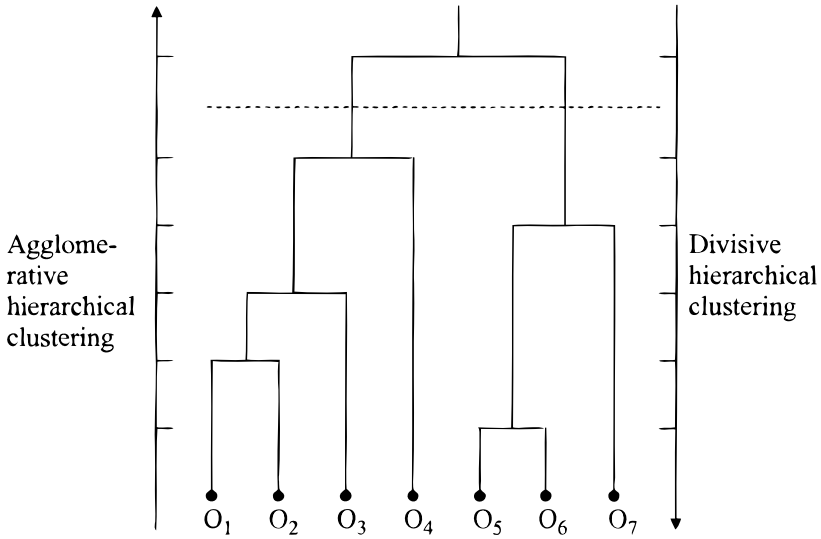
Tyto metody rozdělují data do *k* shluků, kde parametr *k* je předem znám a platí, že $k \leq n$, kde *n* značí počet všech objektů v databázi. Přičemž každý objekt patří do právě jednoho shluku a každý shluk obsahuje alespoň jeden objekt. Využívají techniku *iterativní relokace*, která se snaží rozdělení vylepšit přesouváním objektů mezi shluky [2].

Nejznámějšími zástupci těchto metod jsou algoritmy *K-Means* a *K-Medoids*. Algoritmus *K-Means* bude dále popsán v kapitole 4.1.1.

3.2.2 Hierarchické metody

Vytvářejí stromovou hierarchii shluků. Algoritmy pro hierarchické shlukování využívají 2 základních přístupů: zdola nahoru (agglomerative) nebo shora dolů (divisive). Metody shora dolů začínají na 1 shluku a postupně jej rozdělují na jednotlivé podshluky. Metody zdola nahoru považují nejprve každý objekt za samostatný shluk, poté postupně slučují 2 nejbližší shluky.

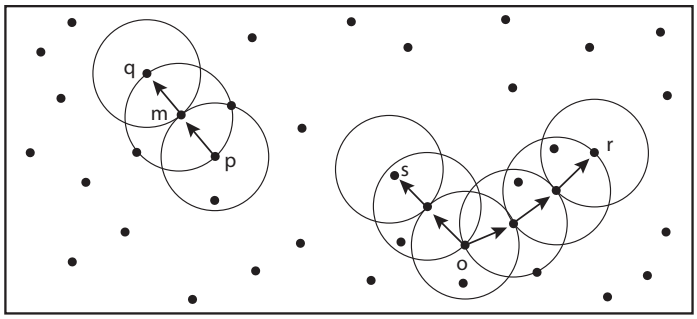
Výsledek hierarchických metod bývá často vizualizován pomocí binárního stromu nebo pomocí tzv. dendrogramu. Dendrogram se velice podobá stromu, na ose y však udává vzdálenost mezi jednotlivými shluky, dendrogram lze vidět na následujícím obrázku (Obrázek 3.1).



Obrázek 3.1: Dendrogram (převzato z [3])

3.2.3 Metody založené na hustotě

Tvoří shluky různých tvarů. Základní myšlenkou těchto metod je postupné zvětšování shluku, pokud se v okolí hraničního bodu nachází dostatečně veliké množství bodů. Taková metoda pak může být použita k odstranění šumu (odlehlých hodnot), či nalezení shluků libovolného tvaru [2]. Na takovémto principu funguje i metoda DBSCAN.



Obrázek 3.2: DBSCAN (převzato z [2])

3.2.4 Metody založené na mřížce

Mapují objekty do n-rozměrné mřížky, která rozděluje prostor na konečný počet buněk (někdy se taky označuje jako kvantizovaný prostor). Operace shlukování pak probíhají nad touto mřížkou. Výhodou těchto algoritmů je velmi krátká doba zpracování.

Mezi mřížkové algoritmy patří například algoritmus STING (viz. [2] a [3]), WaveCluster (popsán v [2], [3] a [1]) .

3.2.5 Metody založené na modelech

Metody založené na modelech se snaží optimalizovat shodu mezi datovou množinou a nějakým matematickým modelem, tzn., snaží se nalézt takové shluky, které by co nejvíce odpovídaly danému modelu. Tyto metody jsou nejčastěji založeny na tom, že data jsou generována na základě nějaké složené pravděpodobnostní distribuční funkce [1]. Také se často ubírají k automatickému určení počtu shluků založenému na standardní statistice, přičemž počítají se šumem a odlehlými hodnotami a tak poskytují robustní shlukovací metody [2].

3.2.6 Shlukování vícedimenzionálních dat

Je velice důležité, protože některé aplikace mohou obsahovat vysoké množství dimenzí (proměnných). Narůstající počet dimenzí může znamenat, že ne všechny budou v datech zaznamenány, čímž roste řídkost dat a některé dimenze se stávají pro danou dolovací úlohu irelevantními. Tím je ovlivněn i výpočet vzdálenosti mezi objekty, který tak ztrácí svůj význam a hustota objektů je také velice nízká. Z toho důvodu se ke shlukování tohoto druhu dat využívají algoritmy odlišné od těch, které jsme si už uvedli, nebo je potřeba provést speciální přípravu shlukovaných dat.

Metody transformace rysů zmenšují počet dimenzí, při zachování vzdálenosti objektů. Avšak tímto způsobem se z dat neodstraní žádné atributy, což může být velmi problematické, zejména pokud je příliš mnoho z těchto atributů irelevantních.

Odlišným přístupem je *výběr podmnožiny atributů*. Při použití takovýchto metod dochází skutečně k redukci dat, při níž jsou irelevantní atributy odstraněny. Problémem je určit, které atributy

se budou odstraňovat. K tomuto účelu se nejčastěji využívá strojového učení s učitelem, nebo je možné využít metod analýzy entropie.

Algoritmy založené na *shlukování podprostoru* využívají poznatku, že odlišné podprostory obsahují jiné zajímavé shluky. Problémem těchto metod je nalezení správného podprostoru. Mezi jejich zástupce patří i algoritmus CLIQUE a PROCLUS, popsán v [3] a [2].

Mezi algoritmy vhodné pro shlukování vícedimensionálních dat patří i algoritmus O-Cluster, který je dále popsán v kapitole 4.2.

3.3 Validace shluků

Metody shlukování se snaží nalézt v datech množiny, v nichž si jsou objekty co nejvíce podobné a objekty mezi nimi by se měly podobat co nejméně. Problémem hodnocení kvality těchto množin (shluků) je fakt, že nemůžeme, tak jako u klasifikace, provést validaci založenou na testovací množině a následně zhodnotit, jestli algoritmus daný objekt správně zařadil do stanovené třídy.

Kvalitu výsledku shlukovacích algoritmů lze posuzovat pomocí některých ze tří základních kritérií. Patří zde externí, interní a relativní kritéria. *Externí kritéria* porovnávají výslednou strukturu shluků s externí strukturou, která naznačuje, jak by shluky měly správně vypadat. *Interní kritéria* ohodnocují strukturu shluků přímo bez potřeby externí informace, přičemž pracují se vzdálenostními maticemi. *Relativní kritéria* porovnávají výsledek s jiným shlukovacím algoritmem nebo stejným algoritmem s odlišnými vstupními parametry.

3.3.1 Externí kritéria

Porovnávají předem známé rozdělení **P** s **C**. Kde **C** je struktura shluků, která je výsledkem shlukovacího algoritmu provedeného na datové množině **X**. Dle externích kritérií pak porovnáváme **P** a **C**. Uvažujme-li body $\mathbf{x}_i$ a $\mathbf{x}_j$ náležící **X**, mohou nastat celkem 4 případy rozdělení těchto bodů mezi **P** a **C**:

1. Oba body náleží stejnému shluku v **C** a stejné kategorii v **P**.
2. Oba body náleží stejnému shluku v **C**, ale každý jiné kategorii v **P**.
3. Každý náleží jinému shluku v **C**, ale oba jsou ve stejné kategorii v **P**.
4. Každý náleží jinému shluku v **C** a také jiné kategorii v **P**.

Pro každý z těchto případů sečteme všechny dvojice, pro které tento případ nastal. Pro případ 1 označíme tento součet jako a, pro případ 2 jako b atd. Pak $a + b + c + d = M$, kde M je množství všech možných dvojic bodů nad **X** a $M = N(N-1)/2$. Hodnotu podobnosti **P** a **C** pak určíme podle některého z následujících vztahů:

Randův index:

$$R = \frac{a + d}{M} \tag{3.23}$$

Jaccardův koeficient:

$$J = \frac{a}{a+b+c} \tag{3.24}$$

Folkes-Mallowsův index:

$$FM = \sqrt{\frac{a}{a+b}\frac{a}{a+c}} \tag{3.25}$$

Výsledná hodnota náleží do intervalu <0, 1>, čím vyšší je, tím se **P** a **C** více podobají a můžeme tedy i tvrdit, že shlukovací algoritmus byl úspěšnější.

3.3.2 Interní kritéria

Jsou vhodná pro hierarchické shlukovací algoritmy. Výsledkem výpočtu je hodnota označovaná jako CPCC (Cophenetic correlation coefficient). CPCC určuje stupeň podobnosti mezi vzdálenostní maticí $\mathbf{P} = \{p_{ij}\}$ a maticí $\mathbf{Q} = \{q_{ij}\}$, jejíž hodnoty značí vzdálenost objektů dle nadřazených shluků, ve kterých byly body poprvé sloučeny.

Takováto definice matice **Q** se obtížně chápe, proto si uvedeme malý příklad. Jestliže budeme uvažovat uvedený dendrogram (Obrázek 3.1), pak v prvku matice, který odpovídá vzdálenosti objektů O_5 a O_6 uvedeme hodnotu 1, kterou můžeme vyčíst z vertikální osy. Ze stejného důvodu pak pro objekty O_1 a O_4 platí vzdálenost 5. Stejná hodnota pak platí i pro vzdálenost O_2 a O_3 od objektu O_4.

Nechť μ_P a μ_Q jsou průměrnými hodnotami matic **P** a **Q**. CPCC se pak vypočítá dle následujícího vztahu

$$CPCC = \frac{\frac{1}{M}\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} p_{ij}q_{ij} - \mu_P\mu_Q}{\sqrt{\left(\frac{1}{M}\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} p_{ij}^2 - \mu_P^2\right)\left(\frac{1}{M}\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} q_{ij}^2 - \mu_Q^2\right)}} \tag{3.26}$$

Hodnota CPCC leží mezi -1 a 1. Pokud se blíží k hodnotě 1, pak značí vyšší kvalitu vytvořených shluků.

3.3.3 Relativní kritéria

Externí a interní kritéria vyžadují statistické testování, které se může stát výpočetně náročným. Relativní kritéria tyto nároky eliminují a zaměřují se na porovnání výsledků generovaných různými shlukovacími algoritmy nebo stejným algoritmem, ale s různými vstupními parametry [3]. U algoritmů založených na rozdělování jsme tak schopni z několika běhů vybrat ten, který dává nejlepší výsledky a určit tak nejlepší hodnotu vstupního parametru *K*. U hierarchických metod zase nalézáme místo, ve kterém bude nejpříhodnější rozdělit dendrogram.

3.3.3.1 Validační indexy

Jedním z možných kritérií jsou i validační indexy. V případě, kdy se pomocí nich snažíme nalézt nejvhodnější místo pro rozdělení dendrogramu u hierarchických shlukovacích metod, označujeme

tyto indexy jako „stop pravidla“ (stopping rules). Více těchto indexů je popsáno v [3], my zde zmíníme postup pro výpočet Dunnova indexu. Dunnův index se snaží identifikovat shluky, které jsou kompaktní a dobře oddělené [3]. Uvažujme shluky C_i a C_j, oddělenost shluků pak určíme pomocí vzdálenosti mezi shluky, která se rovná nejmenší možné vzdálenosti mezi 2 body z uvažovaných shluků:

$$D(C_i, C_j) = \min_{\mathbf{x}\in C_i, \mathbf{y}\in C_j} D(\mathbf{x}, \mathbf{y}) \tag{3.27}$$

Kompaktnost v tomto případě reprezentuje průměr shluku, což je v tomto největší vzdálenost mezi 2 body, které mu náleží:

$$diam(C_i) = \max_{\mathbf{x},\mathbf{y}\in C_i} (\mathbf{x}, \mathbf{y}) \tag{3.28}$$

Nyní už se dostáváme ke konečnému výpočtu Dunnova indexu, pro nějž platí vztah:

$$Du(K) = \min_{i=1,\dots,K} \left(\min_{j=i+1,\dots,K} \left(\frac{D(C_i, C_j)}{\max_{l=1,\dots,K} diam(C_l)} \right) \right) \tag{3.29}$$

Kde K značí počet shluků. Dunnův index zpravidla počítáme vícekrát, tj. jednou pro každý použitý algoritmus (pro různá nastavení algoritmu), získané hodnoty pak porovnáme. Čím vyšší hodnotu index má, tím kvalitnější je i výsledek shlukovacího algoritmu. Nicméně tento index je hodně citlivý na šum a odlehlé objekty. Tím, že se průměr shluku počítá jako maximální vzdálenost mezi dvěma obsaženými body, může jedna odlehlá hodnota celý shluk výrazně ovlivnit. Řešením tohoto problému může být jiný výpočet průměru shluku.

Kromě validačních indexů se do relativních kritérií řadí i další metody, většinou jsou však závislé na zvolené shlukovací metodě.

4 Oracle data mining (ODM)

Oracle dataming (ODM) je placenou součástí databázových serverů firmy Oracle od verze 9iR2. V době psaní této práce je k dispozici verze 11g, proto se všechna následující tvrzení a informace vztažené k ODM a tomuto databázovému serveru budou týkat této verze. K jednotlivým funkcím ODM je možné přistupovat pomocí PL/SQL nebo JAVA API, které je implementací standartu JDM[1].

Dolování se provádí přímo na databázovém serveru Oracle, takovéto řešení eliminuje veliké přesuny dat. ODM nabízí jednak celou řadu metod pro předzpracování dat, jednak i mnoho dolovacích algoritmů. Z těchto algoritmů nás budou zajímat algoritmy pro shlukování. Ty jsou v ODM celkem dva a to *Enhanced K-Means (Hierarchical K-Means)* a *Orthogonal Partition Clustering (O-Cluster)*. Oba tyto algoritmy jsou hierarchické. Algoritmu Enhanced K-Means se bude dále věnovat kapitola 4.1, algoritmu O-Cluster kapitola 4.2.

4.1 Enhanced K-Means

Enhanced K Means je jedním ze shlukovacích algoritmů poskytovaných v ODM. Jedná se o hierarchické rozšíření algoritmu K-Means.

4.1.1 Obecný algoritmus K-Means

K-Means je algoritmus založený na rozdělování. Poskytovanou množinu dat rozdělí do *K* shluků, kde *K* je předem známé celé číslo. Pracuje iteračně ve 2 základních krocích, v prvním přiřadí každý objekt k centrálnímu bodu, ke kterému má nejblíže. Ve druhém kroku přepočítá souřadnice těchto centrálních bodů na jejich průměrné hodnoty ve shluku. Tento postup se opakuje tak dlouho, dokud nepřestane docházet k přesunu bodů mezi shluky.

Algoritmus K-Means:

Vstup: K: Počet shluků, X: Množina s daty

Výstup: C: Množina shluků

Postup:

1. Náhodně zvol *K* centrálních bodů.
2. Každý objekt z X přiřaď k nejbližšímu centrálnímu bodu.
3. Přepočítej souřadnice centrálních bodů.
4. Pokud se souřadnice centrálních bodů nezměnily, ukonči výpočet. Jinak pokračuj od 2.

[1] JSR73: Data Mining API - http://jcp.org/en/jsr/detail?id=73

4.1.2 Enhanced K-Means v prostředí Oracle

Algoritmus K-Means je velice citlivý na volbu počátečních centrálních bodů. Pokud jsou tyto náhodně určené počáteční body blízko ke konečnému řešení, je pravděpodobné, že K-Means skutečně správně určí střed shluku, jinak vede k nesprávným výsledkům. Kvůli náhodné volbě počátečních bodů K-Means negarantuje vždy stejný výsledek shlukování [4].

Tyto i ostatní problémy algoritmu *K-Means* se snaží odstranit celá řada jeho modifikací. Jednou z nich je i hierarchický algoritmus *Enhanced K-Means* (dále jen EKM), který je použit v ODM. Tento algoritmus bývá někdy také označován jako *Hierarchical K-Means*. Avšak i hierarchických algoritmů vycházejících z K-Means je více a je tedy nutné, abychom je mezi sebou nezaměňovali. Algoritmus, který implementuje Oracle, je například odlišný od hierarchického K-Means definovaného v [4].

Algoritmus EKM pracuje jako hierarchický algoritmus metodou shora dolů, který v jednom ze svých kroků využívá K-Means. V jednom okamžiku je vždy dělen jeden uzel stromu na dva nové, vzniklý strom tak nemusí být vyvážený. To, o který uzel půjde, závisí na nastavení, může se dělit buď uzel reprezentující nejméně homogenní shluk, nebo uzel reprezentující shluk s nejvíce objekty. Po každém tomto rozdělení se určí a iterativně přepočítají centrální body nově vzniklých shluků. K tomuto účelu se použije algoritmus K-Means, jehož výpočet skončí nejpozději v okamžiku, kdy je dosažen maximální (předem daný) počet iterací. Celý výpočet se ukončí, jakmile počet uzlů ve stromu dosáhne čísla *K*.

Tento algoritmus vrací pro každý shluk kromě centrálního bodu a ohraničujícího tělesa také histogramy a další informace. Podobná „rozšíření“ lze nalézt i jinde, například mu lze zadat, aby místo Euklidovské vzdálenosti používal vzdálenost kosinovou. Pokud bychom se zaměřili pouze na „jádro“ tohoto algoritmu, mohl by jeho popis vypadat například takto:

Algoritmus Enhanced K-Means:

Vstup: K:počet shluků, I:počet iterací, X:datová množina, S:kritérium pro rozdělení

Výstup: Stromová struktura shluků

Postup:

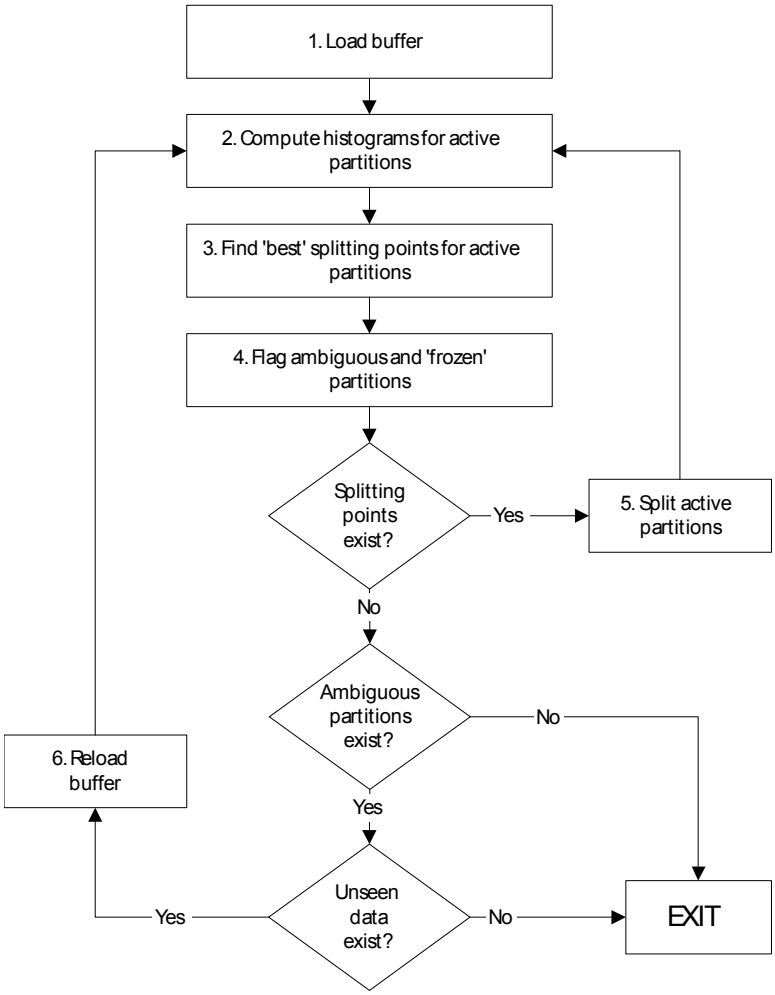
1. Uvažuj celou datovou množinu **X** jako 1 shluk a označ ho, jako vybraný.
2. Rozděl vybraný shluk na 2 shluky.
3. Na nově vzniklé shluky aplikuj algoritmus K-Means, zastav se po **I** iteracích.
4.
    a. Pokud je počet shluků roven K, přejdi na krok 5.
    b. Jinak vyber shluk pro další dělení v závislosti na kritériu **S** a přejdi na krok 2.
5. Ukonči výpočet a vrať hierarchii shluků.

Tímto přístupem ke K-Means se vyhneme potřebě běhu více modelů K-Means a dostáváme výsledky, jejichž konzistence je vyšší, než při použití tradičního K-Means [5].

4.2 Orthogonal Partition Clustering (O-Cluster)

Algoritmus O-Cluster je rekurzivní hierarchický algoritmus využívající nepravidelnou mřížku. Pracuje s omezenou pamětí, proto je vhodný i pro shlukování velikých objemů dat, zároveň je schopen zpracovávat vícedimensionální data.

O-Cluster je postaven na metodách, které byly představeny v algoritmu OptiGrid (popsán v [6]). Jeho princip spočívá v postupném dělení hyperprostoru hyperplochami, čímž jej rozděluje a vytváří tak nepravidelnou pravoúhlou n-rozměrnou mřížku. Po každém takovém rozdělení se shluk rozdělí na 2 nové shluky a tím se buduje binární strom, jedná se tedy o hierarchickou metodu shora dolů.



Obrázek 4.1: Diagram algoritmu O-Cluster (převzato z [6])

Obrázek 4.1 ilustruje jednotlivé kroky algoritmu O-Cluster. Tyto kroky si zde vysvětlíme:

1. Provede se vzorkování dat, poté se náhodným vzorkem naplní vstupní buffer. V případě, že se do vstupního bufferu vejdou všechna data, je naplněn celý.
2. Pro každý nerozhodnutý[2] listový shluk se nalezne vhodná projekce. Ta se aplikuje na jeho objekty a z výsledků se následně vypočítá histogram. Lokální maximum v histogramu se označuje jako *vrchol*, lokální minimum jako *údolí*. Vrchol histogramu pak značí místo, ve kterém je vyšší hustota objektů než v okolí. V údolí je tomu naopak.
3. Pro každý histogram se algoritmus snaží nalézt nejlepší protínající hyperplochu, ta prochází některým bodem spadajícím do údolí v histogramu. Dělící hyperplocha pak bude procházet bodem s nejnižší hustotou. Nejvhodnější je nalézt co nejmenší údolí obklopené co nejvyššími vrcholy. Ne vždy se ovšem takový bod podaří nalézt, algoritmus se tedy rozhoduje pomocí následujícího testu[3]:

$$\chi^2 = \frac{2(pozorovaná - očekávaná)^2}{očekávaná} \qquad (4.1)$$

   Kde pozorovaná hodnota se rovná počtu objektů v údolí histogramu, očekávaná je průměrem mezi počtem objektů v údolí a v nižším vrcholu. Pokud je hodnota vyšší než daný práh[4], algoritmus považuje tento bod za dělící. Takovýchto bodů může být nalezeno i více, dělí se ovšem vždy pouze v tom nejlépe hodnoceném.
4. V tomto kroku se rozhoduje, zda se bude algoritmus pokoušet daný shluk v budoucnu znova dělit. Znovu se provede test z kroku 3, ale uvažuje se nižší prahová hodnota. Pokud je test splněn, shluk je označen jako nerozhodnutý (ambiguous). V opačném případě je brán jako zmražený (frozen) a už se s ním nebudou provádět další operace. Objekty, které mu náleží, jsou následně v bufferu označeny ke smazání. Takový shluk pak bude v konečném výsledku ležet v listu stromu.
5. Pokud byl v kroku 3 nalezen dělící bod, je jím v tomto kroku vedena hyperplocha a vzniknou tak 2 nové shluky. Po rozdělení shluku se rekurzivně zavolá funkce provádějící krok 2.
6. Pokud jsou všechny shluky označeny jako zmražené, nebo již byly všechny objekty zpracovány, algoritmus se zde ukončí. Jinak jsou z bufferu odebrány objekty, které byly označeny k odebrání. Prázdná místa jsou poté zaplněna objekty, které náleží do nerozhodnutých shluků. Poté se opět pokračuje v kroku č. 2.

[2] Po prvním kroku je to pouze jeden, obsahující veškerá data ze vstupního bufferu.
[3] Vychází ze standardního chí kvadrát testu. Vzorec je pouze zjednodušen pro tento konkrétní případ.
[4] Současná implementace uvažuje hodnotu $\chi^2 = 3{,}843$ [6].

5 Systém vyvíjený na FIT

Systém vyvíjený na FIT je systémem pro dolování z dat, na němž pracují studenti v rámci svých diplomových a bakalářských prací. Vývoj systému započal v roce 2006 Ing. Doležal. Výsledkem jeho práce bylo jádro systému umožňující přidávat dolovací moduly [7]. Systém byl dále přepracován na NetBeans platformě s použitím NetBeans Visual Library a pracuje s daty uloženými na databázových serverech Oracle. V současném stavu byl vyvinut pro verzi 10g, nově se bude pracovat s verzí 11g. Tento systém se skládá z jádra a modulů, které ho dále rozšiřují. Z hlediska procesu získávání znalostí z dat jsou v jádře implementovány zejména funkce pro předzpracování. Moduly pak implementují jednotlivé dolovací úlohy a další rozšíření.

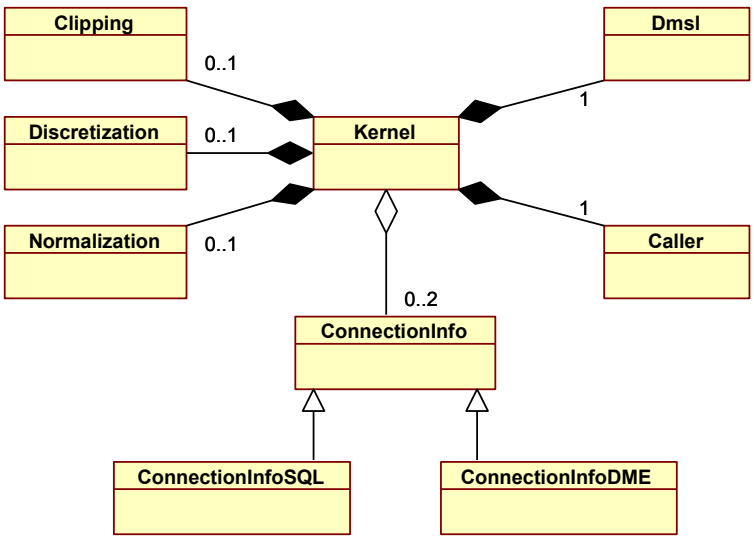
5.1 Současný stav

Systém však zatím nepodporuje všechny typy dolovacích úloh, které jsme si uváděli v kapitole 2.3. Lze na něm však již provádět:

- Dolování asociačních pravidel.
- Klasifikaci rozhodovacím stromem a Bayesovskou klasifikaci.
- Predikci pomocí regrese.

5.1.1 Jádro systému

Tvoří základ celého systému, jeho současnou podobu ovlivnil zejména Ing. Krásný [8], jenž definoval rozhraní pro připojování modulů. Velmi důležitým tahem bylo také použití datových struktur odvozených z DMSL, systém tak tedy pracuje s dolovacími úlohami na úrovni DMSL.

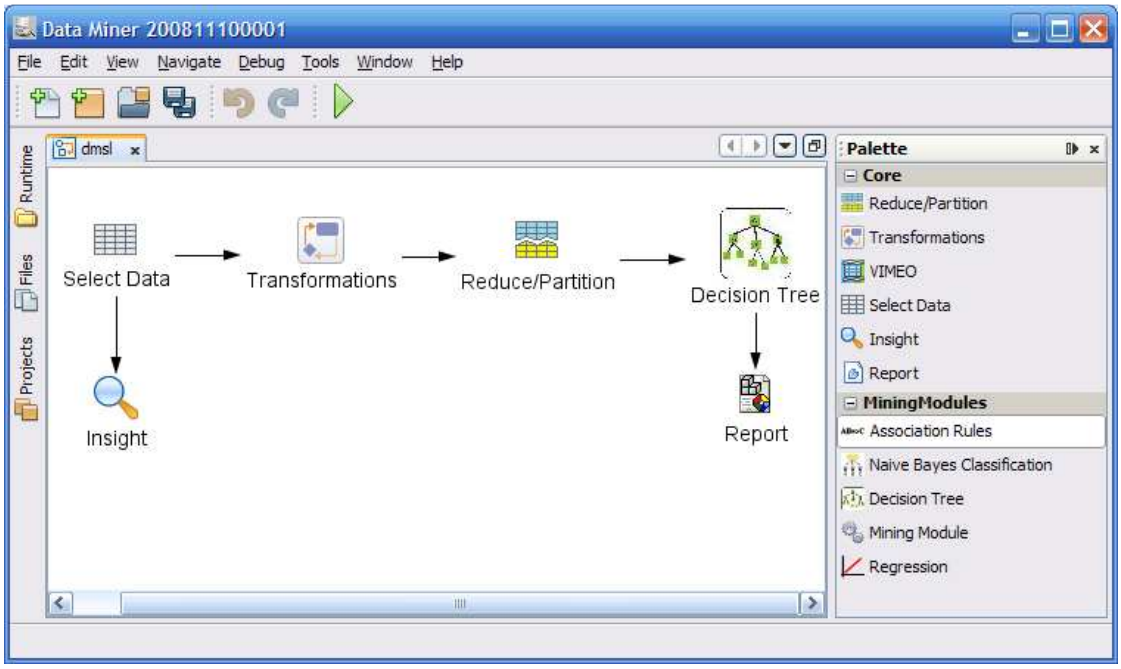


Obrázek 5.1: Jádro systému (převzato z [8])

Na dalších změnách v jádře se podílel Ing. Šebek [7], který dopracoval jádro do současné podoby. Jednalo se především o předělání komponent pro předzpracování dat, vytvoření abstraktní vrstvy pro přístup k datům a řešení vícevláknového běhu aplikace.

5.1.2 GUI

Grafické uživatelské rozhraní systému je ukázáno na následujícím obrázku (Obrázek 5.2). Důležitý je zde především způsob propojování jednotlivých komponent dolovacího procesu.



Obrázek 5.2: GUI systému

Jak lze z obrázku vidět, komponenty jsou umístěny na „pracovní ploše“, na kterou se přidávají z palety (vpravo). Na pracovní ploše jsou dále propojeny v pořadí, ve kterém bude celý dolovací proces probíhat.

5.1.3 Komponenty dolovacího procesu

Jednotlivé komponenty jsou vidět již na předchozím obrázku (Obrázek 5.2) v paletě aplikace, kde jsou rozděleny na moduly jádra a dolovací moduly. Úlohy jednotlivých modulů jsou následující:

- **Select Data** slouží k výběru dat pro dolovací úlohu. Lze pomocí něj však i importovat tabulku ze souboru do databáze.
- **Insight** prohlíží data tak jak vypadají po kroku, na který je připojen. Kromě náhledu do samotné tabulky umožňuje také zobrazení různých statistických přehledů, či počítání korelace mezi dvěma atributy.
- **Vimeo** provádí převážně čistění dat.
- **Transformations**, jak již název napovídá, slouží k transformaci dat.

- **Reduce/Partition** provádí redukci dat vybraným způsobem. Lze ji také využít pro rozdělení dat na trénovací a testovací množinu pro metody klasifikace a predikce.

Dalšími komponentami jsou již samotné dolovací moduly provádějící klasifikaci, predikci a dolování asociačních pravidel.

5.1.4 Moduly

Moduly jsou běžnými součástmi moderních systémů, jsou odděleny od jádra a jsou na něm nezávislé. Pro moduly však musí být definováno rozhraní, pomocí kterého probíhá komunikace s jádrem. V našem případě se jedná o abstraktní třídu `MiningPiece`, od níž všechny moduly povinně dědí. Jakoukoliv třídu od ní dědící lze pak připojit do grafu dolovacího procesu. Každý implementovaný modul musí poskytovat formulář pro zadávání parametrů specifických pro danou úlohu a panel pro prezentaci výsledků [8].

5.2 NetBeans platforma

NetBeans je projekt firmy Sun Microsystems. Základem celého projektu je NetBeans platforma, na níž je postaveno i vývojové prostředí NetBeans IDE. Jedná se o tzv. framework s modulární architekturou, který je určen k usnadnění výstavby desktopových aplikací. Tato platforma sama využívá Swing UI, avšak zjednodušuje některé použití, například připojování akcí k položkám v menu, tvorbu klávesových zkratek apod.

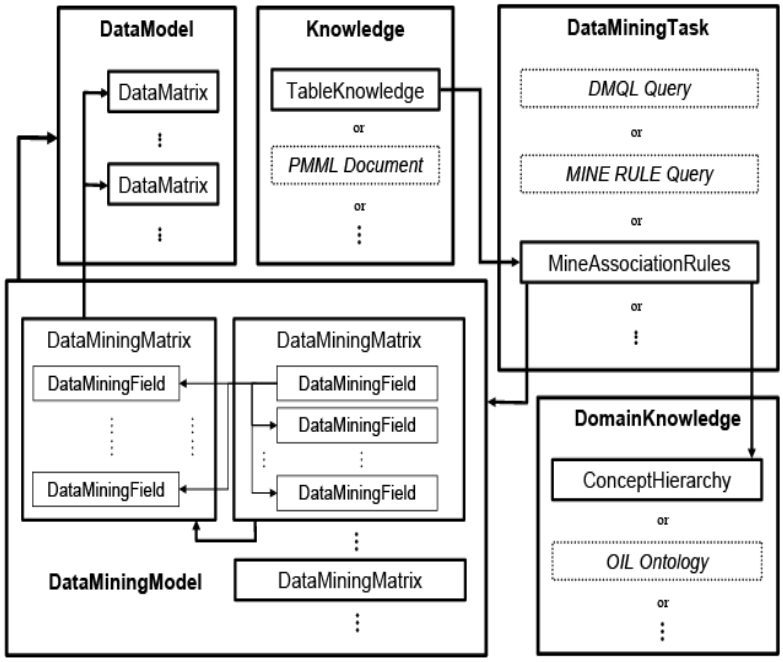
NetBeans Visual Library je grafická knihovna určená k vizualizaci a grafově orientovanému modelování. Na této knihovně je založeno i rozhraní systému pro dolování z dat vyvíjeného na FIT.

5.3 DMSL

DMSL (Data Mining Specification Languge) je jazykem založeným na XML, sloužícím k popisu celého procesu dolování dat včetně výsledných znalostí. DMSL definuje 5 hlavních elementů, které hrají hlavní role v procesu dolování dat [9]:

1. *Datový model* slouží k popisu vstupních dat.
2. *Model dolování dat* slouží k popisu dat poté, co byly provedeny potřebné transformace.
3. *Doménová znalost* umožňuje popsat znalosti o dolovaných datech, které mohou být využity při dolování.
4. *Dolovací úloha* specifikuje typ dolovací úlohy.
5. *Znalost* reprezentuje výstup dolovací úlohy.

Závislosti těchto elementů ilustruje následující obrázek 5.3.



Obrázek 5.3: Závislosti DMSL elementů (převzato z [9])

6 Návrh a implementace modulu shlukové analýzy

Na vývoji shlukovacího modulu jsme pracovali ve dvojici, mým spolupracovníkem byl Bc. Martin Hlosta [11]. Naše role při tomto vývoji však byly jasně definovány tak, aby se co nejméně překrývaly a abychom mohli pracovat co nejvíce samostatně. Naším společným úkolem byl návrh shlukovacího modulu včetně implementace důležitých rozhraní a abstraktních tříd. Také však i implementace finálních tříd, jejichž úkolem je začlenění modulu do systému a základní funkcionalita. Mé samostatné úkoly pak byly

- začlenění algoritmů, které využívají ODM (O-Cluster a Enhanced K-Means),
- vizualizace výsledků shlukování a
- návrh a implementace modulu, který bude provádět validaci a porovnání výsledků (viz. Kapitola 8)

Společnou část popisuje kapitola 6 s výjimkou části věnované začlenění algoritmů z ODM. Zbylé kapitoly se věnují samostatné práci.

6.1 Začlenění modulu do systému

Aby bylo možné modul do systému začlenit, je potřeba implementovat třídu *MiningPiece*. Ta obsahuje metody, které by se daly rozdělit do více skupin. Jsou zde metody komunikující s jádrem, metody uživatelského rozhraní, metody volané po událostech nastávajících při práci s grafem dolovacích komponent. Třída *MiningPiece* tedy sjednocuje komunikaci s rozsáhlejším systémem.

Dalším krokem pro začlenění modulu do systému je upravení záznamu v souboru *layer.xml*. Právě díky tomuto souboru se nakonec systém dozví, které moduly lze v diagramu popisujícím dolovací proces připojit na vstup našeho a které moduly lze naopak připojit na jeho výstup.

6.2 Návrh modulu shlukové analýzy

Modul jsme se snažili od počátku navrhnout tak, aby byl snadno udržovatelný a lehce rozšiřitelný. Třídy a balíčky jsme navrhovali takovým způsobem, aby měly co nejvyšší kohezi a byly co nejslaběji provázány. Rovněž jsme při návrhu využívali standardní návrhové vzory.

Modul je organizován do několika balíčků, výchozí balíček *clusteringmodule*[5] obsahuje pouze třídu *ClusteringModule* dědící od *MiningPiece*. Skutečná logika modulu se ukrývá v jeho

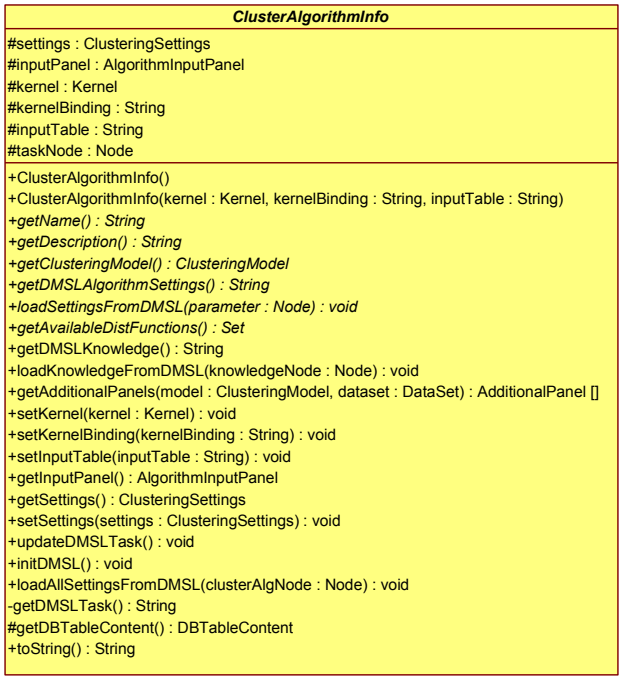
[5] Celým názvem *cz.fit.vutbr.dataminer.module.clusteringmodule*, pro přehlednost zde uvádíme pouze zkrácený název.

podbalíčcích, které si zde popíšeme. Záměrně zde budeme uvádět pouze důležité třídy a jejich metody, které hrají významnou roli z pohledu návrhu modulu. Kompletní popis všech balíčků a tříd je uveden v dokumentaci. Mezi tyto balíčky patří

- balíček *api*, jenž obsahuje důležitá rozhraní a třídy, které je nutné implementovat pro začlenění nového algoritmu do modulu,
- balíček *algorithms*, který obsahuje třídy a balíčky reprezentující jednotlivé algoritmy,
- balíček *data*, s jehož pomocí jsou v rámci modulu reprezentována data,
- balíček *distfunctions*, který obsahuje vzdálenostní funkce,
- balíček *gui* obsahuje uživatelské rozhraní modulu,
- balíček *model* reprezentuje výsledek shlukovacího algoritmu a
- balíček *utils* poskytuje metody, které jsou v různých částech programu používány velmi často, většinou jsou tyto metody volány přes statický kontext.

6.2.1 Balíček api

Shlukovací modul umožňuje uživateli zvolit si algoritmus, který chce použít. Pro realizaci této funkčnosti jsme zvolili návrhový vzor strategy, který se v balíčku api uplatňuje hned několikrát. U algoritmů samotných, je onou třídou, od níž všechny konkrétní algoritmy dědí třída *ClusterAlgoritmInfo*.

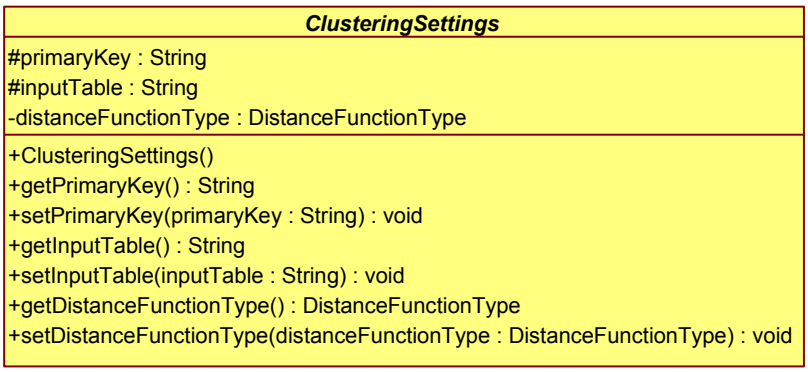


Obrázek 6.1: Třída ClusterAlgoritmInfo

Obrázek 6.1 ukazuje, jak tato třída vypadá. Na první pohled lze vidět, že je poměrně hodně rozsáhlá, pro algoritmy v systému představuje tato třída to samé, co *MiningPiece* pro moduly. Její rozsáhlost plyne z toho, že každý algoritmus má odlišné vstupní požadavky. S těmi také souvisí odlišný zápis do

jeho vyhrazené části v DMSL, proto musí být tento zápis a také čtení implementováno pro každý algoritmus samostatně. Tato třída také poskytuje metody pro spouštění shlukovací úlohy a obdržení jejich výsledků.

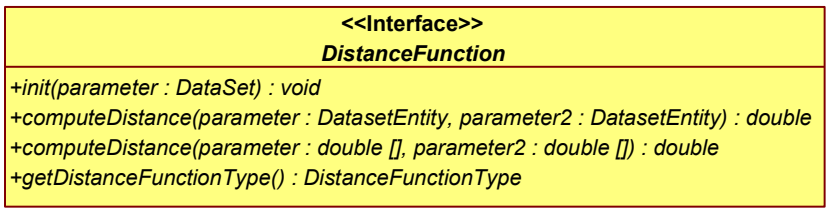
Každý algoritmus má jiné vstupní požadavky. K tomu, aby s těmito požadavky, nebo také nastaveními, bylo možné pracovat i obecněji, je v balíčku *api* definována další abstraktní třída – *ClusteringSettings* (Obrázek 6.2).



Obrázek 6.2: Třída ClusteringSettings

Tato třída v základu poskytuje algoritmům nastavení, která jsou pro každý z nich shodná (tzn. typ vzdálenostní funkce, tabulku se vstupními daty, primární klíč). Další parametry algoritmů (počet shluků, citlivost, atd.) se objevují dále v jejích konkrétních implementacích. S těmito nastaveními také souvisí jejich získávání a zobrazování uživateli, proto je v tomto balíčku také definována abstraktní třída pro vstupní panel, která pro ně poskytuje dvojici getter-setter a také booleovskou metodu pro validaci vstupu.

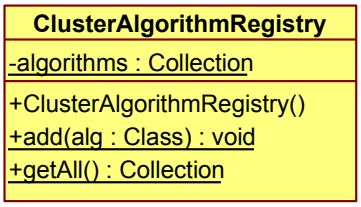
Kromě nových algoritmů je také modul snadno rozšiřitelný i o další typy vzdálenostních funkcí. K realizaci vzdálenostních funkcí je opět užit návrhový vzor strategy. Obrázek 6.3 ukazuje, jak toto rozhraní vypadá, více o vzdálenostních funkcích bude popsáno v kapitole 6.2.4.



Obrázek 6.3: Rozhraní DistanceFunction

6.2.2 Balíček algorithms

Balíček *algorithms* obsahuje další balíčky s realizacemi konkrétních shlukovacích algoritmů. Mimo to také obsahuje 2 důležité třídy. Obrázek 6.4 ukazuje první z nich, třídu *ClusterAlgorithmRegistry*. Právě tato třída shromažďuje informace o dostupných algoritmech.



Obrázek 6.4: Třída ClusterAlgorithmRegistry

Každý algoritmus se nejprve zaregistruje voláním funkce *add(Class)*. Instance zaregistrovaných tříd se pak vytvoří v případě potřeby, tj. při vytvoření nové instance celého modulu.

Druhou třídou tohoto balíčku je *ODMClusteringTask*, a jak její název napovídá, je důležitá pouze pro algoritmy z ODM, důvody pro její zavedení budou vysvětleny v kapitole 6.2.2.1.

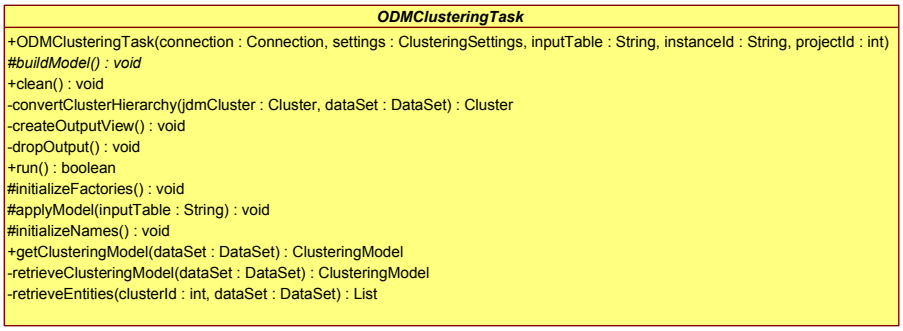
6.2.2.1 Adaptace algoritmů z ODM

S algoritmy z ODM lze pracovat pomocí rozhraní JDM. Kompletní úlohu shlukování lze rozdělit na 2 části:

1. Vytvoření modelu.
2. Aplikování modelu.

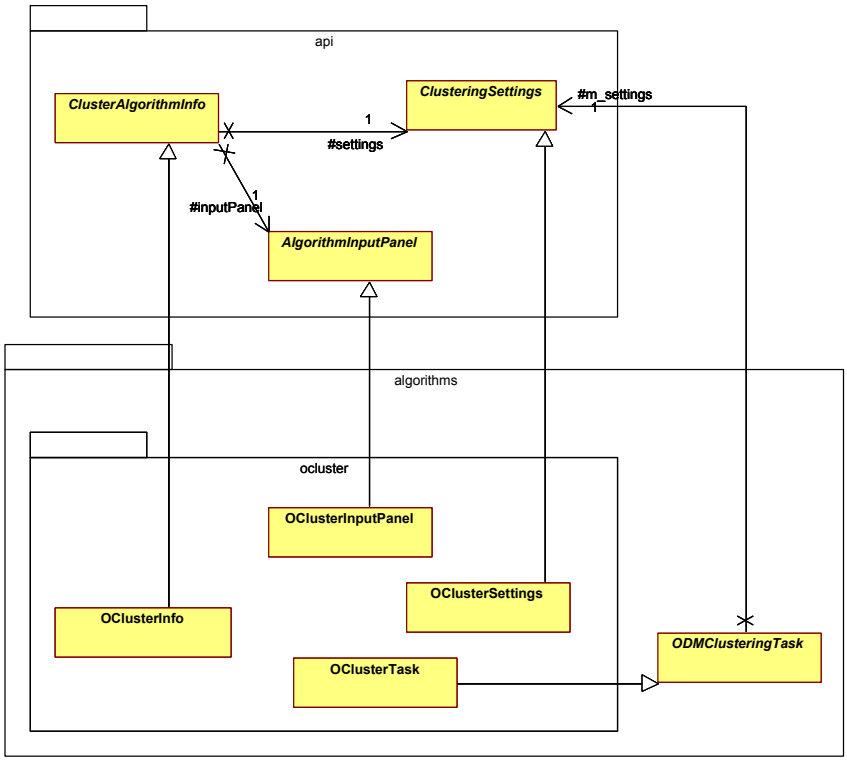
Vytvořený model, dle standartu JDM, obsahuje o shlucích především statistické informace, jako je například počet objektů ve shluku apod. Informace o tom, které objekty shluk obsahuje, se z tohoto modelu nedozvíme, obsahuje však pravidla, pomocí nichž můžeme zjistit, ke kterému shluku objekt přiřadit. Nejjednodušší způsob, jakým lze takovou informaci získat je ale aplikace vytvořeného modelu na vstupní tabulku. Výsledkem této operace je nová tabulka, obsahující 3 sloupce, kterými jsou: cizí klíč do vstupní tabulky, identifikace shluku a sloupec udávající pravděpodobnost, že daný objekt k danému shluku patří. Podle nejvyšší pravděpodobnosti příslušnosti ke shluku lze pak tuto tabulku transformovat tak, aby obsahovala dvojice id-shluk.

Celý proces shlukování v ODM se u různých algoritmů liší pouze v části týkající se vytvoření modelu. Zbytek je u obou poskytovaných algoritmů stejný. Proto je v balíčku *algorithms* zavedena abstraktní třída *ODMClusteringTask* (Obrázek 6.5), která provádí úkoly, jež jsou pro tyto algoritmy společné. Dále pak definuje abstraktní metodu *buildModel()*, kterou si pak každá odvozená třída implementuje sama podle sebe.



Obrázek 6.5: Třída ODMClusteringTask

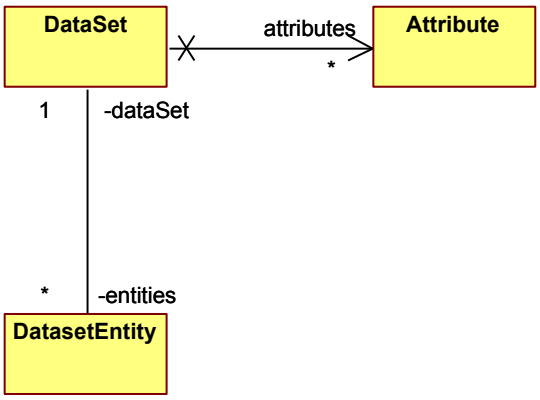
Implementace jednotlivých algoritmů pak spočívá v implementaci abstraktních tříd z balíčku *api* a třídy *ODMClusteringTask*. Obrázek 6.6 znázorňuje vztahy mezi třídami z jednotlivých balíčků pro případ algoritmu O-Cluster.



Obrázek 6.6: Vztahy mezi třídami balíčků api a cluster

6.2.3 Balíček data

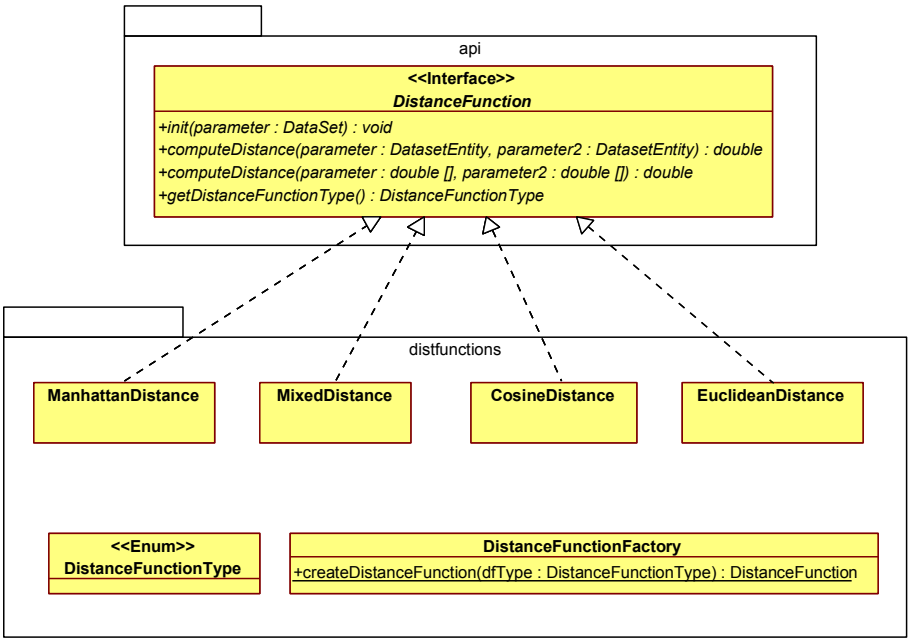
Slouží k reprezentaci datové množiny v rámci shlukovacího modulu. Je založen na třech třídách: *DataSet* reprezentuje celou datovou množinu, *Attribute* uchovává údaje o atributech a *DatasetEntity* reprezentuje jeden objekt z datové množiny (Obrázek 6.7).



Obrázek 6.7: Třídy obsažené v balíčku data

6.2.4 Balíček distfunctions

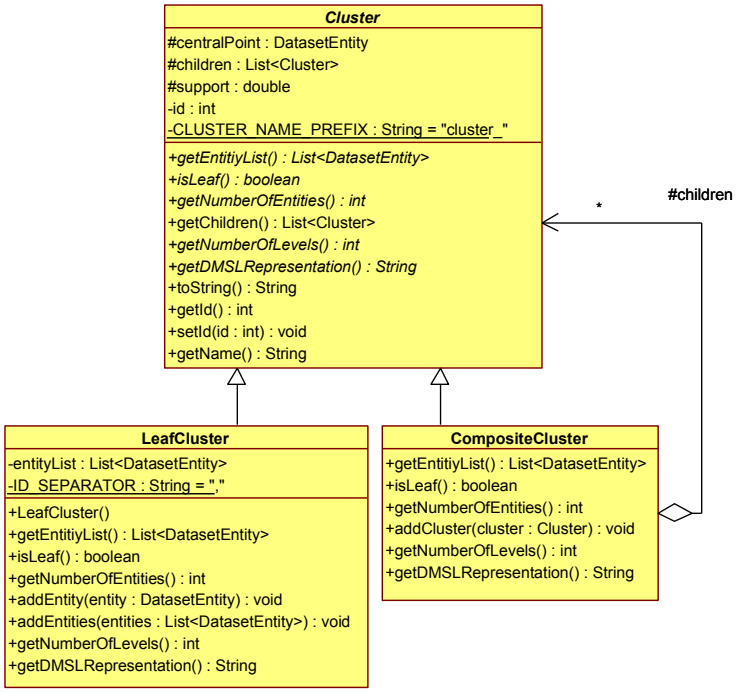
O návrhu rozhraní pro vzdálenostní funkce jsme se lehce zmínili již v kapitole 6.2.1. Balíček *distfunctions* obsahuje již jednotlivé implementace vzdálenostních funkcí založených na rozhraní *DistanceFunction* (Obrázek 6.3). Mimo to je zde ještě třída, která řeší vytváření nových instancí těchto funkcí – třída *DistanceFunctionFactory* (Obrázek 6.8).



Obrázek 6.8: Vztah balíčků api a distfunctions

6.2.5 Balíček model

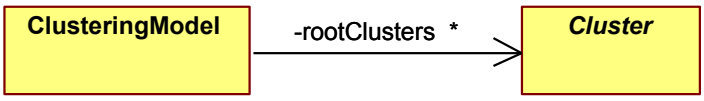
Jak již název napovídá, právě tento balíček reprezentuje výsledek shlukovacího algoritmu, čili v tomto případě znalost. Reprezentace nalezených shluků je pro systém velice důležitá, proto jsme kladli veliký důraz na její správný návrh. Úkolem zde bylo navrhnout model tak, aby byl schopen správně reprezentovat výsledky jak nehierarchických, tak i hierarchických algoritmů. Při návrhu tříd, představujících jeden shluk, jsme tak využili návrhový vzor composite (Obrázek 6.9).



Obrázek 6.9: Třídy představující shluky

Třída *CompositeCluster* tak tedy představuje nelistový shluk, třída *LeafCluster* listový. Pokud chceme získat objekty obsažené ve shluku, zavoláme metodu *getEntityList()*, listový shluk při tomto volání pouze vrátí kolekci, kterou si udržuje. Nelistový vrátí spojení kolekcí všech svých potomků ve stromu. Podobným způsobem zde pracují i některé další metody.

Další důležitou třídou balíčku model je třída *ClusteringModel*. Ta reprezentuje celý výsledek, obsahuje metody pro získání shluků podle typu apod. Důležité je, že si shluky ukládá v kolekci *rootClusters*, povoluje tak více než jeden kořenový shluk (Obrázek 6.10). Toho využíváme u nehierarchických algoritmů, u kterých tato kolekce obsahuje všechny shluky. Ty pak tedy uvažujeme zároveň jako kořenové i listové. Přestože takové metody v modulu zatím nemáme, dá se tento model teoreticky využít i pro metody jejichž výsledkem je více než jen jeden strom.



Obrázek 6.10: Uložení shluku ve třídě ClusteringModel

6.3 Záznamy v DMSL

DMSL dokument má z hlediska dolovacích modulů 2 zásadní významy:

1. Slouží jako úložiště pro nastavení modulu.
2. Slouží jako úložiště pro získanou znalost.

Pro první možnost slouží v DMSL dokumentu element *DataMiningTask*, pro druhou *Knowledge*. Poměrně důležitý je zde fakt, že se tyto elementy mohou v rámci DMSL dokumentu

vyskytovat vícekrát. Z pohledu celého systému to pro nás pak znamená, že můžeme v rámci jednoho projektu používat více dolovacích modulů. Oba tyto elementy definují povinný atribut *name* do kterého každý modul vloží svoji identifikaci.

6.3.1 Nastavení dolovacího modulu

Nastavení dolovacího modulu se odlišují podle typu vybraného algoritmu, nicméně i tak lze nalézt vlastnosti, které se vyskytují u každého z nich. Z pohledu ukládání do DMSL tak můžeme hovořit o takových elementech a atributech, které jsou společné pro všechny algoritmy a o takových, které se vyskytují jen u některých z nich.

K uložení nastavení využijeme zmíněný element *DataMiningTask*, do nějž definujeme vlastní element shlukové analýzy.

```
<!ELEMENT ClusterAnalysis
(UseMatrix,(OCLUSTER|DBSCAN|EKMEANS|OPTICS|DENCLUE))>
<!ATTLIST ClusterAnalysis algorithm CDATA #REQUIRED>
<!ATTLIST ClusterAnalysis distanceFunction CDATA #REQUIRED>
<!ATTLIST ClusterAnalysis primaryKey CDATA #REQUIRED>
```

Element *ClusterAnalysis* tak tedy definuje, který z výčtu elementů v něm může být vnořen, zároveň také definuje 3 povinné atributy: *algoritm*, *distanceFunction*, *primaryKey*. Tyto atributy představují právě ona společná nastavení. Mimo ně ještě obsahuje vnořený element UseMatrix, který obsahuje referenci na vstupní tabulku.

Specifická nastavení pak obsahuje další vnořený element. U algoritmů využívajících ODM se jedná o elementy *OCLUSTER* a *EKMEANS*, jejichž definice vypadá následovně:

```
<!ELEMENT OCLUSTER (max_number_of_clusters, max_buffer_size, sensitivity)>
<!ELEMENT EKMEANS (max_number_of_iterations, minimal_error_tolerance,
number_of_bins, block_growth, minimal_percentage_attribute_support,
max_number_of_clusters, split_criterion)>
<!ELEMENT max_buffer_size (#PCDATA)>
<!ELEMENT max_number_of_clusters (#PCDATA)>
<!ELEMENT max_number_of_iterations (#PCDATA)>
<!ELEMENT minimal_error_tolerance (#PCDATA)>
<!ELEMENT number_of_bins (#PCDATA)>
<!ELEMENT block_growth (#PCDATA)>
<!ELEMENT minimal_percentage_attribute_support (#PCDATA)>
<!ELEMENT split_criterion (#PCDATA)>
<!ELEMENT sensitivity (#PCDATA)>
```

Vidíme, že některé elementy, třeba *max_number_of_clusters*, se mohou vyskytovat ve více algoritmech. Význam důležitých parametrů je následující:

- *max_buffer_size* udává maximální velikost vstupního bufferu u algoritmu O-Cluster,
- *sensitivity* představuje citlivost, čím je vyšší, tím více nalezne algoritmus shluků,
- *max_number_of_clusters* značí u algoritmu O-Cluster maximální počet nalezených listových shluků, u Enhanced K-Means se jedná o přesný počet (parametr K),
- *max_number_of_iterations* udává, po kolika iteracích se má zastavit hledání středových bodů,
- *minimal_error_tolerance* má vliv na kvalitu a rychlost vytváření modelu, čím vyšší je její hodnota, tím rychleji je model vytvořen,
- *minimal_percentage_attribute_support* se vztahuje k jednotlivým atributům, pokud je procento nechybějících hodnot menší, než zadané číslo, je atribut brán jako irelevantní,
- *split_criterion* představuje kritérium pro výběr děleného shluku. Možnostmi jsou zde:
    - *Variance* pro rozdělení nejméně kompaktního shluku a
    - *Size* pro rozdělení největšího shluku.

Příklad nastavení algoritmu O-Cluster pak může vypadat následovně:

Příklad:

```
<DataMiningTask language="XML" name="DM545_ClusteringModule3">
  <ClusterAnalysis algorithm="O-Cluster" distanceFunction="EUCLIDEAN"
  primaryKey="ID">
    <UseMatrix matrixRef="DM545_Select1"/>
    <OCLUSTER>
      <primary_key>ID </primary_key>
      <max_number_of_clusters>10 </max_number_of_clusters>
      <max_buffer_size>50000 </max_buffer_size>
      <sensitivity>0.5 </sensitivity>
    </OCLUSTER>
  </ClusterAnalysis>
</DataMiningTask>
```

Jelikož je toto nastavení závislé na vybraném shlukovacím algoritmu, musí se každý z nich o čtení a zápis starat sám. Tuto činnost provádějí v modulu metody potomků třídy *ClusterAlgoritmInfo*. Uložení nastavení do DMSL je nutné provést jednak při vložení modulu na „pracovní plochu“ systému, pak také vždy, když jsou provedeny změny ze strany uživatele. Jejich načtení je nutné provést vždy před spuštěním shlukovací úlohy. Při načítání se nejprve identifikuje použitý algoritmus

pomocí atributu *algorithm*, poté v načítání pokračuje třída zastupující příslušný algoritmus. Výsledkem operace načtení je vždy instance potomka třídy *ClusteringSettings*.

6.3.2 Znalost dolovacího modulu

Výsledky shlukování jsou v rámci modulu reprezentovány vždy stejným způsobem bez ohledu na to, který algoritmus úlohu provedl. Díky tomu je způsob ukládání znalosti do DMSL jednotnější oproti nastavením jednotlivých algoritmů. Znalost je v modulu reprezentována třídami balíčku *model* (kapitola 6.2.5). Při ukládání modelu do DMSL jde tedy v podstatě o převedení instancí tříd tohoto balíčku do XML a následné vložení do elementu *knowledge*. Pro znalost jsou tedy definovány následující elementy:

```
<!ELEMENT Model (ModelSettings, Clusters)>

<!ELEMENT ModelSettings EMPTY>
<!ATTLIST ModelSettings algorithmName CDATA #REQUIRED>
<!ATTLIST ModelSettings buildTable CDATA #REQUIRED>

<!ELEMENT Clusters (Cluster*)>
<!ATTLIST Clusters numberOfLevels CDATA #IMPLIED>
```

Všechny elementy, které popisují znalost, jsou obsaženy v elementu *Model*. *ModelSettings* představují obecný popis výsledného modelu, jejich parametr *buildTable* obsahuje název vstupní tabulky. Jednotlivé shluky jsou pak obsaženy uvnitř elementu *Clusters* a jsou definovány následovně:

```
<!ELEMENT Cluster (Cluster+|Entities)>
<!ATTLIST Cluster id CDATA #REQUIRED>
<!ATTLIST Cluster isLeaf (true|false) #REQUIRED>

<!ELEMENT Entities EMPTY>
<!ATTLIST Entities identifiers CDATA #REQUIRED>
```

V kapitole Balíček model6.2.5 jsme si uvedli, že shluky v systému odpovídají návrhovému vzoru composite. Při návrhu záznamu v DMSL to pak znamená, že element cluster musí umožňovat, aby do něj byl vložen další element cluster.

Z uvedeného DTD lze vidět, že shluk může obsahovat také element *Entities* s atributem *identifiers*. Tento atribut obsahuje čárkami oddělený seznam hodnot primárního klíče ze vstupní tabulky. Pomocí něj je možné ve výsledku přiřadit objekt ze vstupní tabulky ke správnému shluku.

7 Prezentace výsledků shlukování uživateli

Tato kapitola popisuje, jakým způsobem prezentujeme výsledky shlukování uživateli. Obrázky v této kapitole jsou přímo grafy získané z vytvořeného modulu, pokud není explicitně uvedeno jinak.

Existuje celá řada způsobů, kterými můžeme znázornit výsledky shlukování, avšak mnoho z nich závisí na obsahu shlukovaných dat a také použitém algoritmu (např. tvorba evolučního stromu). Způsobů, kterými lze znázornit obecný výsledek, je již méně. Při vizualizaci velice záleží na tom, co vlastně chceme znázornit. Pro tento modul jsem se tak rozhodl implementovat 3 typy vizualizace, a to:

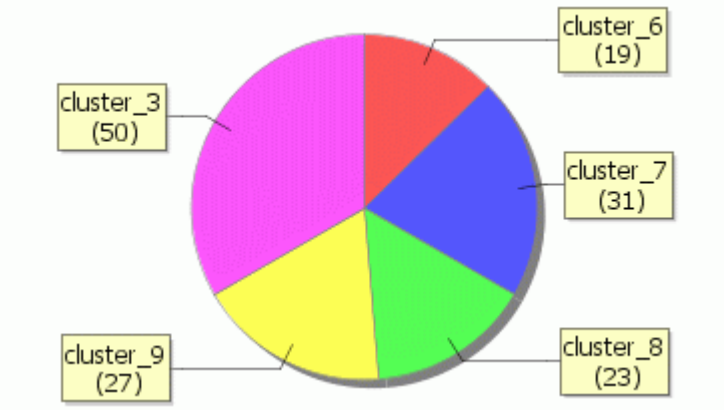
1. vizualizaci modelu jako celku, porovnání shluků mezi sebou,
2. vizualizaci založenou na nominálních proměnných, která znázorňuje zastoupení nominálních hodnot v jednotlivých shlucích a
3. vizualizaci založenou na intervalových proměnných, díky níž si dokážeme při malém množství atributů udělat představu o tvaru nalezených shluků, jejich velikosti a vzdálenostmi mezi nimi.

Tyto 3 typy vizualizace se zobrazují vždy v dialogovém okně pro prohlížení výsledků, každá z nich je zobrazena v panelu pod vlastní záložkou. Každý algoritmus však ještě může volitelně poskytovat další panely s vizualizací pro něj specifickou. Ze současně implementovaných algoritmů to tak činní algoritmus OPTICS.

7.1 Vizualizace celého modelu

Ačkoliv nadpis obsahuje slovo vizualizace, ve skutečnosti se v tomto přídně jedná spíše o způsoby poskytování informací o modelu uživateli. Tyto informace zpravidla prezentujeme v grafické podobě, to však není podmínkou.

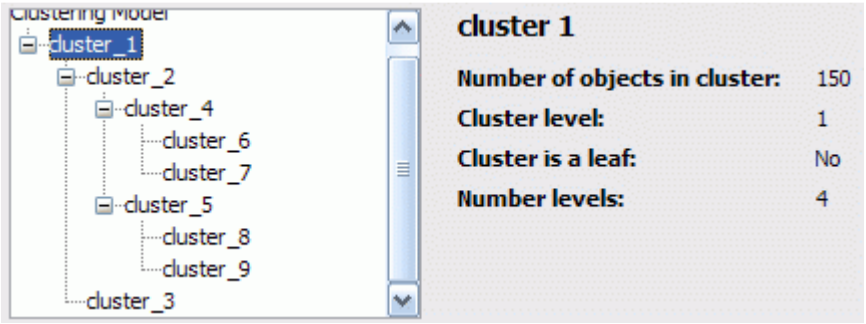
Uživateli můžeme poskytnout porovnání v závislosti na velikosti shluku. Obrázek 7.1 znázorňuje, jak toto porovnání vypadá v případě použití koláčového grafu, doplněného o další popisky. U uvedeného příkladu lze hned na první pohled vidět, že největší shluk, je cluster_3.



Obrázek 7.1: Porovnání velikostí shluků za pomoci koláčového grafu

V našem modulu využíváme tento graf pro porovnání listových shluků, nebo všech shluků v případě algoritmů, které nepracují hierarchicky. Tento graf je dále také interaktivní, pokud uživatel najede kurzorem myši nad jednu z výsečí, zobrazí se mu další informace, například kolik procent objektů ze vstupní datové množiny přísluší právě tomuto shluku.

Další informací, kterou můžeme uživateli prezentovat je hierarchické uspořádání shluků (Obrázek 7.2).



Obrázek 7.2: Strom shluků vytvořený hierarchickým algoritmem

Jak je z obrázku vidět, k zobrazení této hierarchie v modulu používáme běžný formulářový prvek (JTree). Výhoda takového přístupu spočívá v tom, že nabízí uživateli možnost přepínání mezi jednotlivými shluky a tím pádem i zobrazení dodatečných informací.

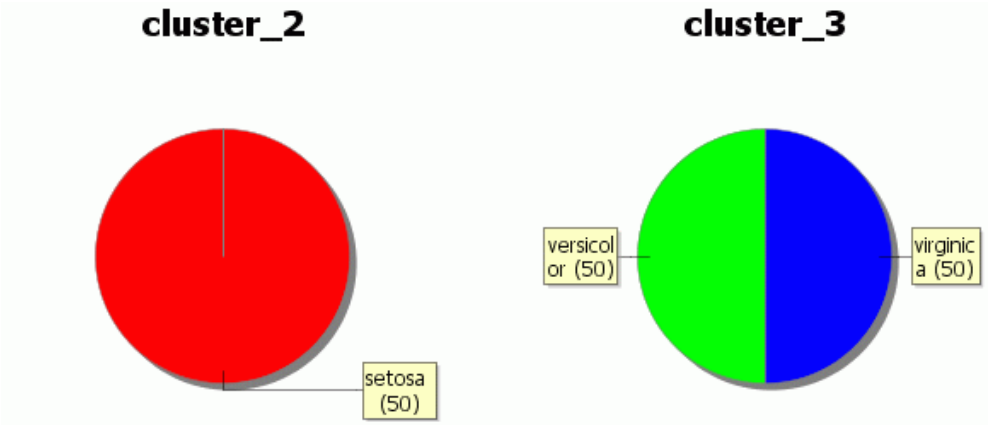
Kromě koláčového grafu a hierarchického přehledu dále modul nabízí další přehled, udávající informace o celkovém počtu shluků, počtu listových shluků, apod.

7.2 Vizualizace založená na nominálních proměnných

K tomu, abychom dobře porozuměli vizualizaci nominálních proměnných, je potřeba si je porovnat s proměnnými intervalovými. Co se týče vizualizace, mají intervalové proměnné jednu velikou výhodu. Pokud bychom měli datovou množinu o jedné, dvou, či třech intervalových proměnných, dokázali bychom si při nanesení hodnot do bodového grafu lehce představit, jaké jsou vzdálenosti

mezi jednotlivými objekty. U nominálních proměnných to však takto jednoduché není. Proto k jejich znázornění typicky nepoužíváme způsoby, které nám poskytují přehled o vzdálenosti mezi jednotlivými objekty, ale zaměřujeme se spíše na zastoupení nominálních hodnot v jednotlivých shlucích.

V našem shlukovacím modulu je k tomuto účelu využit opět koláčový graf. Zobrazujeme jich vedle sebe hned několik (v závislosti na počtu shluků). Obrázek 7.3 ukazuje, jak může toto znázornění vypadat v případě 2 shluků nalezených v Iris Data Setu[6] z UCI Repository [11], tuto datovou množinu budeme používat i v dalších příkladech.



Obrázek 7.3: Zastoupení hodnot nominálních proměnných ve shlucích

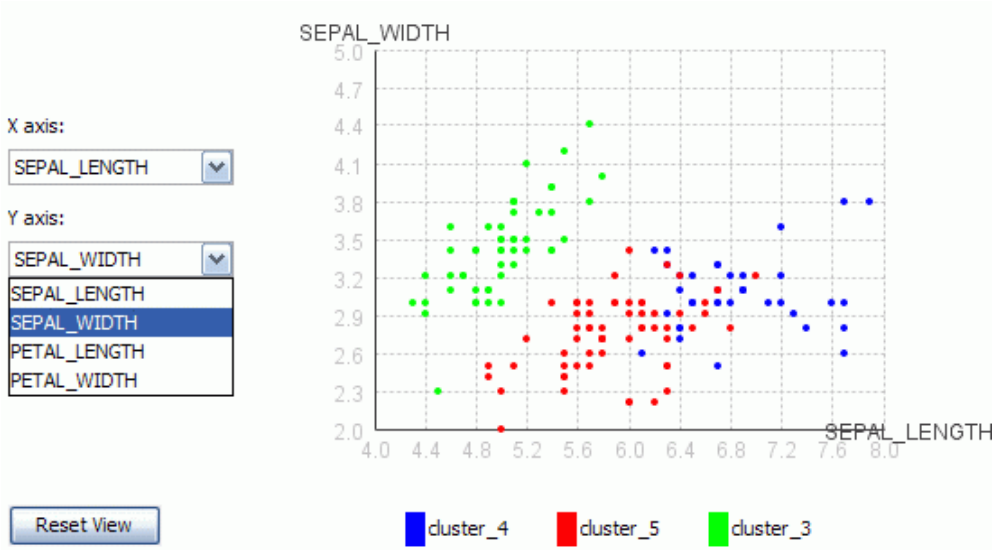
I podle tohoto typu vizualizace dokážeme odhadnout kvalitu vytvořených shluků. V případě koláčových grafů bychom nejraději viděli kolečka stejné barvy, tj. v každém shluku 100% zastoupení jedné z hodnot, tak jak je tomu například u shluku cluster_2 z obrázku.

K vykreslování koláčových grafů využívá modul knihovnu JFreeChart [12].

7.3 Vizualizace založená na intervalových proměnných

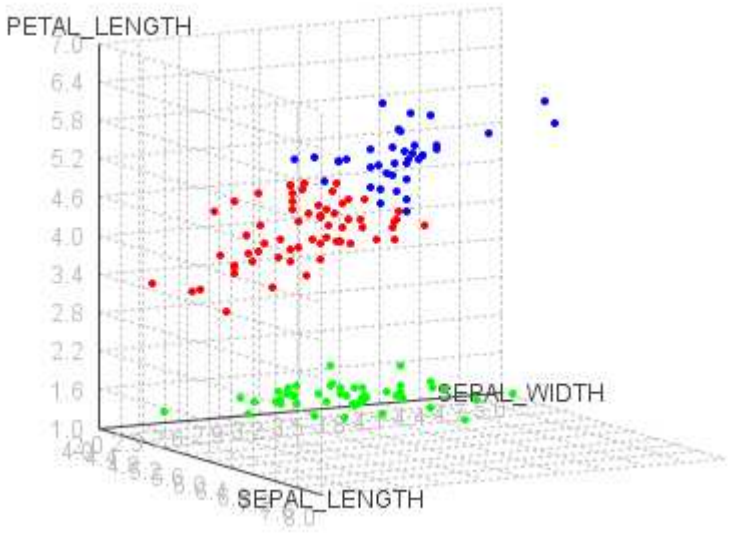
Tento typ vizualizace poskytuje snad nejlepší způsob, jak uživateli skutečný výsledek přiblížit. Hodnoty intervalových proměnných naneseme do bodového grafu a jednotlivé shluky odlišíme barvou. Ze všech dostupných intervalových proměnných pak umožníme uživateli, aby si zvolil, na kterou osu chce nanést hodnoty vybrané proměnné.

[6] Datová množina obsahující záznamy o 150 jedincích rostliny kosatec. Pro 3 druhy této rostliny obsahuje záznamy o délce a šířce okvětních a kališních lístků.



Obrázek 7.4: Vizualizace intervalových proměnných pomocí 2D bodového grafu

Avšak ani tento způsob grafického znázornění není dokonalý. Pokud se podíváme na graf nahoře (Obrázek 7.4), může se nám zdát, že se shluky cluster_4 a cluster_5 překrývají. Pokud ovšem přidáme třetí osu, může vypadat situace jinak.



Obrázek 7.5: Vizualizace intervalových proměnných pomocí 3D bodového grafu

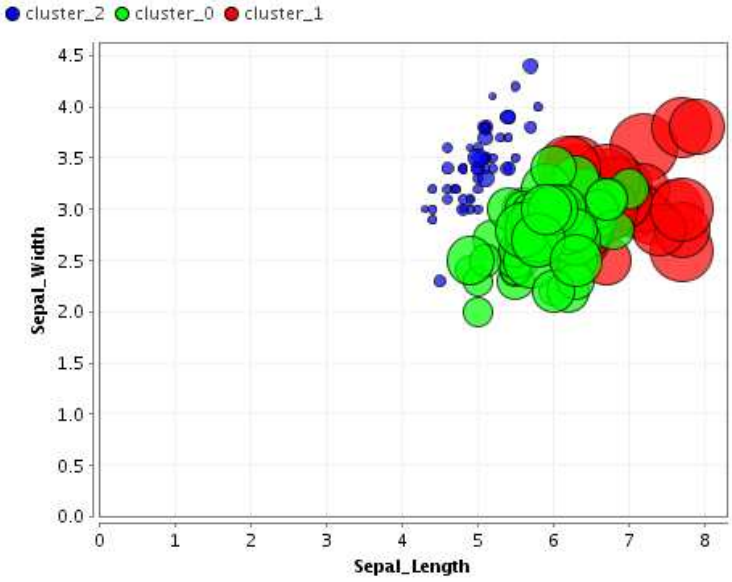
Obrázek 7.5 prezentuje stejný výsledek pomocí trojrozměrného grafu a o nalezených shlucích si tak můžeme udělat lepší přehled.

V obou těchto případech můžeme říct, že barva vytváří další rozměr grafu. Pomocí dvojrozměrného grafu, tak vlastně prezentujeme uživateli ve skutečnosti 3 rozměry, kde onen třetí rozměr je příslušnost bodu do shluku. K vizualizaci založené na intervalových proměnných využívá modul knihovnu JMathPlot [13].

7.3.1 Další formy vizualizace

Bodové grafy se stávají méně přehlednými, pokud počet dimenzí datové množiny vysoce přesahuje počet dimenzí v grafu. Existují tak i další typy grafů, které se snaží tento problém řešit jiným způsobem, než přidáním další osy, ty však v modulu už implementovány nejsou.

Jedním z těchto grafů je i bublinkový graf (Obrázek 7.6), který hodnoty dalšího atributu znázorňuje pomocí velikosti vykreslovaného bodu. U tohoto příkladu lze z grafu vyčíst, že na přiřazení odlehlejšího bodu do shluku cluster_2 se velkou měrou podílí hodnota, která je znázorněna právě pomocí velikosti bublinky.



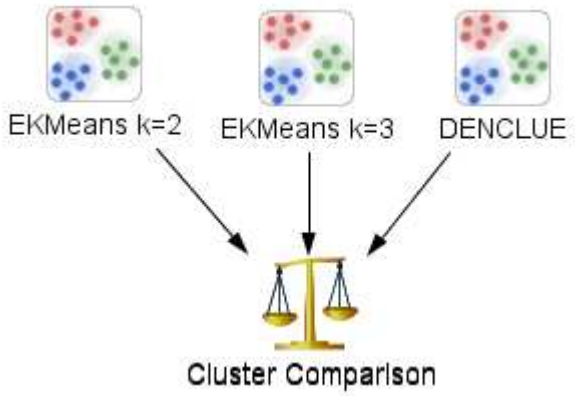
Obrázek 7.6: Bublinkový graf z aplikace Rapid Miner [13]

8 Návrh a implementace porovnávacího modulu

Úkolem tohoto modulu je porovnat výsledky dvou nebo více shlukovacích modulů. Toto porovnání provádíme pomocí tzv. validačních indexů (kapitola 3.3.3.1). Výsledkem těchto metod je reálné číslo, které je relativní, tzn., že podle jeho hodnoty nelze určit, nakolik přesně je výsledek správný. Víme ale, že čím je tato hodnota vyšší, tím byl i výsledek shlukování lepší. Této vlastnosti pak můžeme využít pro porovnání jednotlivých výsledků mezi sebou a právě to provádí porovnávací modul.

8.1 Návrh

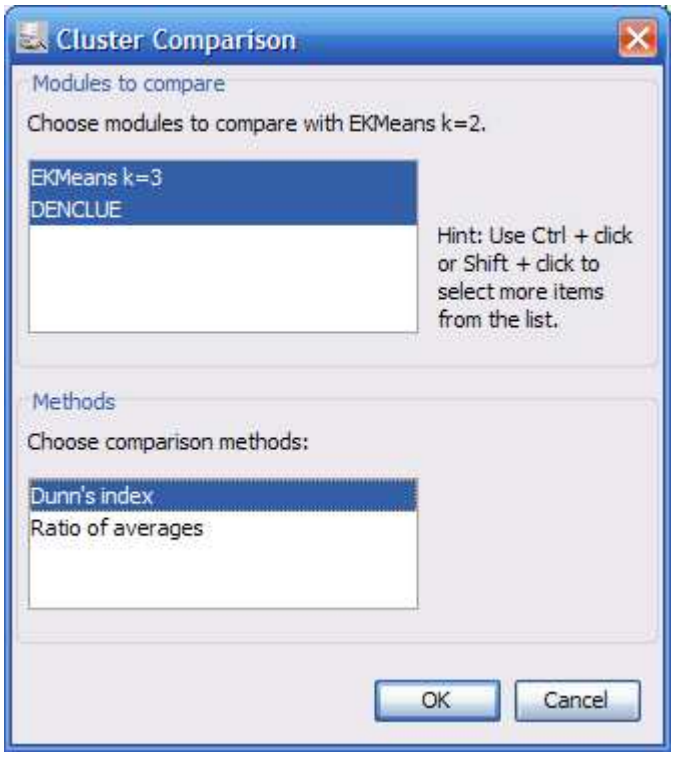
Z pohledu uživatele by bylo nejpřehlednější, pokud bychom mohli vybrat porovnávané moduly tak, že bychom je zapojili na vstup modulu porovnávacího (Obrázek 8.1).



Obrázek 8.1: Porovnávací modul

Tímto způsobem to bohužel řešit nelze, protože je systém navržen tak, aby graf komponent dolovacího procesu tvořil vždy strom. Byla by tedy vyžadována změna v jádře, díky které by komponenty mohly tvořit na ploše acyklický graf.

Po domluvě s vedoucím práce jsem se tedy uchýlil k řešení, kdy se porovnávací modul zapojí pouze na jeden modul shlukovací. Další moduly v grafu jsou poté dohledány pomocí metod, procházející strom dolovacích komponent. Uživatel si pak při otevření dialogu vybere, se kterými z modulů na ploše chce připojený modul porovnat, dále také vybere typ počítaného indexu. U obou těchto nastavení lze vybrat více položek zároveň (Obrázek 8.2).



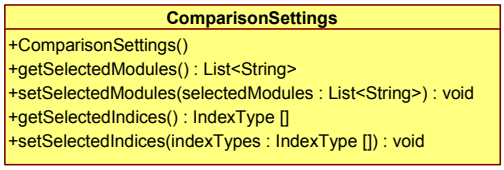
Obrázek 8.2: Dialog s nastaveními porovnávacího modulu

Modul je navržen také tak, aby bylo v budoucnu možné, pokud dojde ke změně v jádře, modul lehce upravit, aby mohl fungovat prvním zmíněným způsobem. Jediné, co bude v takovém případě potřeba, je

1. zbavit se metod, které vyhledávají shlukovací moduly ve stromě a místo nich využít kolekci *sourceConnections* definovanou v třídě *MiningPiece*,
2. odebrat z třídy *ComparisonSettings* kolekci *selectedModules*, která představuje seznam připojených modulů a
3. z třídy *DMSLUtil* se zbavit metod, které souvisejí s ukládáním tohoto seznamu do DMSL.

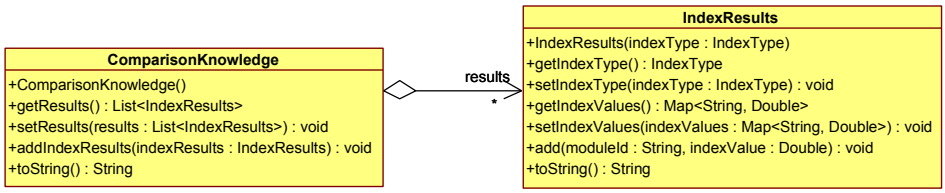
8.1.1 Důležité třídy

I v tomto modulu je potřeba si uchovávat nastavení a výslednou znalost. Nastavení zde představuje třída *ComparisonSettings* (Obrázek 8.3).



Obrázek 8.3: Třída ComparisonSettings

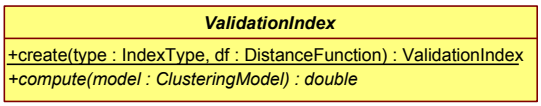
Ta uchovává seznam vybraných modulů pro porovnání a také seznam vybraných validačních indexů. Další důležitou třídou je pak třída *ComparisonKnowledge*, která není až tak jednoduchá, jak by se mohlo na první pohled zdát.



Obrázek 8.4: Třídy ComparisonKnowledge a IndexResults

Obrázek 8.5 znázorňuje třídy *ComparisonKnowledge* a *IndexResults*. Všimněme si, že metoda *getResults()* vrací seznam objektů třídy *IndexResults*. Třída *IndexResults* představuje seznam výsledků pro jeden vypočítaný index, to zde znamená, že jednak poskytuje informaci o typu počítaného indexu a dále pak mapuje každý vypočítaný výsledek na modul.

Nyní už se však dostáváme k samotnému porovnání. Celý tento proces se spouští přes třídu *ComparisonTask*, jejím úkolem je pro každý zvolený index a každý zvolený modul provést výpočet. K reprezentaci jednotlivých typů výpočtu používáme návrhový vzor strategy. Díky tomuto přístupu lze v budoucnu modul snadno rozšířit o další způsoby porovnání.

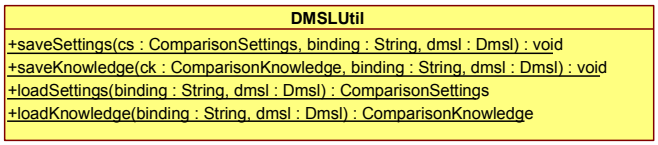


Obrázek 8.5: Abstraktní třída ValidatonIndex

Každý validační index je tak potomkem abstraktní třídy *ValidationIndex* (Obrázek 8.5), a musí poskytovat metodu *compute()*, která jako parametr přebírá vytvořený shlukovací model. Instance jednotlivých validačních indexů vytváříme pomocí statické metody *create()*, ta v závislosti na předaném typu indexu vytvoří potomka tak, že zavolá jeho stejnojmennou metodu, jedná se tak tedy o užití návrhového vzoru factory method. Všimněme si také, že tato metoda přebírá typ vzdálenostní funkce. Při výpočtu indexu tak tedy používáme stejnou vzdálenostní funkci, jaká byla použita při shlukování.

8.1.2 Záznamy v DMSL

Stejně tak, jako je tomu u všech dolovacích modulů, i porovnávací modul musí ukládat svá nastavení a svoji znalost do DMSL. V tomto případě se o všechny tyto operace stará třída *DMSLUtil* (Obrázek 8.6).



Obrázek 8.6: Třída DMSLUtil

Při ukládání nastavení se tak tedy jedná o přepis instance třídy *ComparisonSettings* do podoby XML, stejně tak je tomu i u znalosti v případě třídy *ComparisonKnowledge*.

Nastavení porovnávacího modulu jsou v elementu *DataMiningTask* a definujeme je následovně:

```
<!ELEMENT selected_modules (module*)>
<!ELEMENT module (#PCDATA)>

<!ELEMENT selected_indices (index*)>

<!ELEMENT index EMPTY>
<!ATTLIST index type CDATA #REQUIRED>
```

Z uložených nastavení si tak tedy můžeme vyčíst vybrané moduly a vybrané typy validačních indexů. Elementy *module* obsahují jednoznačné identifikace shlukovacích modulů.

Znalost modulu je uložena v elementu *Knowledge*:

```
<!ELEMENT indices (index+)>

<!ELEMENT index (index_result+)>
<!ATTLIST index type CDATA #REQUIRED>

<!ELEMENT index_result EMPTY>
<!ATTLIST index_result module CDATA #REQUIRED>
<!ATTLIST index_result value CDATA #REQUIRED>
```

Element *indices* obsahuje seznam validačních indexů (element *index*). V tom jsou pak uvedeny pro každý vybraný modul výsledky výpočtu. Příklady záznamů *DataMiningTask* a *Knowledge* mohou vypadat následovně:

Příklad:

```
<DataMiningTask language="XML" name="DM545_ClusterComparison1">
  <selected_modules>
    <module>DM545_ClusteringModule1</module>
    <module>DM545_ClusteringModule2</module>
    <module>DM545_ClusteringModule3</module>
  </selected_modules>
```

```
   <selected_indices>
     <index type="Dunn's index"/>
     <index type="Ratio of averages"/>
   </selected_indices>
</DataMiningTask>

<Knowledge language="XML" name="DM545_ClusterComparison1">
   <indices>
     <index type="Dunn's index">
       <index_result module="DM545_ClusteringModule3"
       value="0.061103691082127626"/>
       <index_result module="DM545_ClusteringModule2"
       value="0.3389086708546395"/>
       <index_result module="DM545_ClusteringModule1"
       value="0.07650633600840401"/>
     </index>
     <index type="Ratio of averages">
       <index_result module="DM545_ClusteringModule3"
       value="0.0662958406483381"/>
       <index_result module="DM545_ClusteringModule2"
       value="0.4513012352319814"/>
       <index_result module="DM545_ClusteringModule1"
       value="0.094362538563885"/>
     </index>
   </indices>
</Knowledge>
```

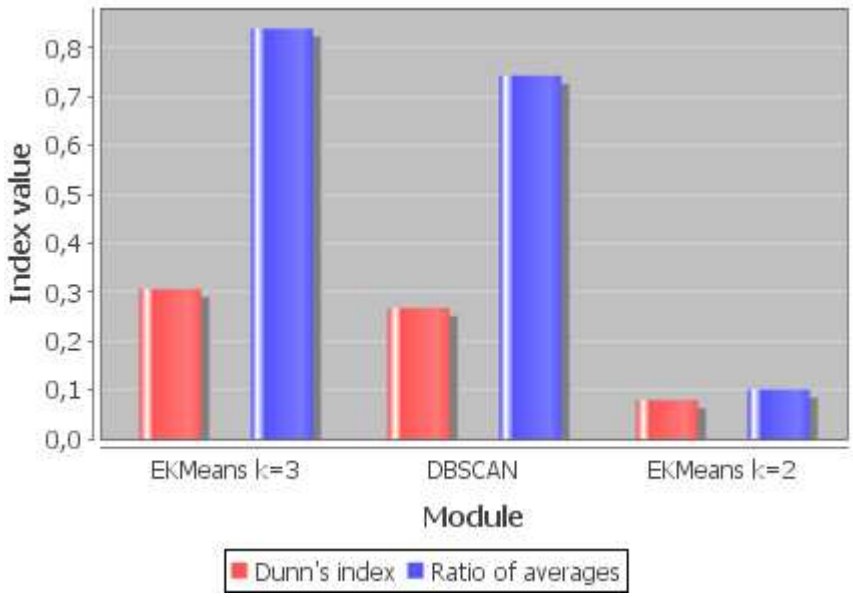
8.2 Prezentace výsledků

Způsobů, kterými uživateli výsledky poskytujeme, je více. V základu mu můžeme nabídnout výsledky v textové podobě, což je také jeden ze způsobů, který tento modul nabízí. Tím druhým a zároveň přehlednějším způsobem je porovnání pomocí grafu. V tomto případě je vhodné použít sloupcový graf. Z takového grafu lze pak lehce vyčíst, kde nám vyšel lepší výsledek. Modul pro jeho vykreslení používá knihovnu JFreeChart [12].

Obrázek 8.7 ukazuje výsledek porovnání vybraných modulů, které hledaly shluky v množině Iris Data Set. Červený sloupec udává hodnotu Dunnova indexu pro daný model, modrý sloupec představuje poměr mezi průměrnou vzdáleností mezi shluky a průměrným rozměrem shluku[7]. Kvalitu

[7] V druhém případě se nejedná o kvalitní metodu porovnání. Čím více shluků je nalezeno, tím poskytuje horší výsledky, často upřednostňuje výsledek s velikým počtem shluků.

modelu tak poznáme podle výšky sloupce (čím vyšší tím lepší). U tohoto příkladu tak dopadlo nejlépe shlukování metodou Enhanced K-Means s počtem listových shluků nastaveným na hodnotu 3.



Obrázek 8.7: Porovnání výsledků pomocí sloupcového grafu

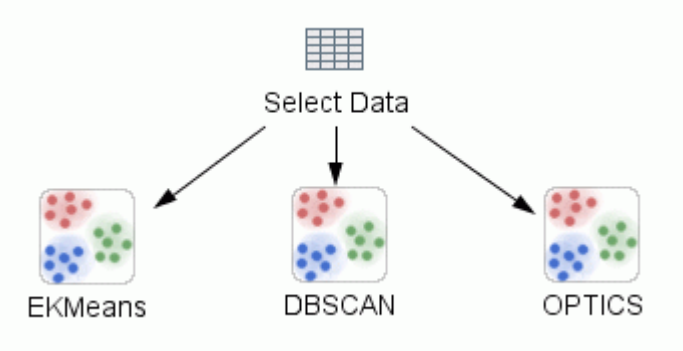
Validační indexy mají však své nedostatky a může se stát, že každý z vybraných určí jiného vítěze. Právě proto tento modul zobrazuje více výsledků v jednom grafu.

9 Používání modulů

V této kapitole si ukážeme, jakým způsobem lze modul používat. Předvedeme, jakým způsobem nastavíme celý proces dolování a jak vytvořit více shlukovacích modelů, které si následně porovnáme. Demonstrujeme tak tedy i činnost porovnávajícího modulu a ověříme kvalitu jeho výsledků. Pro tyto ukázky využijeme vlastní data upravená záměrně tak, aby se ukázaly schopnosti jednotlivých algoritmů. Tato data obsahují pouze 2 relevantní atributy *x* a *y*, abychom byli schopni si podle grafu udělat dobrou představu o nalezených shlucích. Představené úloha bude značně zjednodušená, při reálném dolování bychom navíc prováděli také různé formy předzpracování dat. Použitá data naleznete na datovém nosiči odevzdaném s touto diplomovou prací.

Nejprve importujeme data do databáze, důležité je aby tabulka obsahovala sloupec s unikátními hodnotami, který budeme moci použít jako primární klíč. Po spuštění aplikace Data Miner vytvoříme nový projekt pomocí *File -> New Project*. Vybereme *Data Mining Project* z kategorie *Knowledge Discovery*, svou volbu potvrdíme stisknutím tlačítka *Next*. Dále název projektu zadáme a po potvrzení nastavíme připojení k databázi a naši volbu ukončíme stiskem tlačítka *Finish*. V levém panelu pod kartou *Projects* je zobrazen strom projektů, 2x klikneme na soubor *dmsl.xml*, který patří k nově vytvořenému projektu. Poté se nám zobrazí pracovní plocha a načte se paleta přístupných modulů.

Teď se již dostáváme k vytváření dolovací úlohy. Nejprve přetáhneme na plochu modul *Select Data*. Klikneme na něj pravým tlačítkem a z kontextového menu zvolíme *Open*. Vybereme naší importovanou tabulku a zvolíme všechny sloupce. Poté přetáhneme na plochu 3 instance modulu *Cluster Analysis*. Moduly shlukové analýzy napojíme na *Select Data* stiskem klávesy *Ctrl* a současným tahem myší ze *SelectData* na *Cluster Analysis*.



Obrázek 9.1: Připojení na Select Data

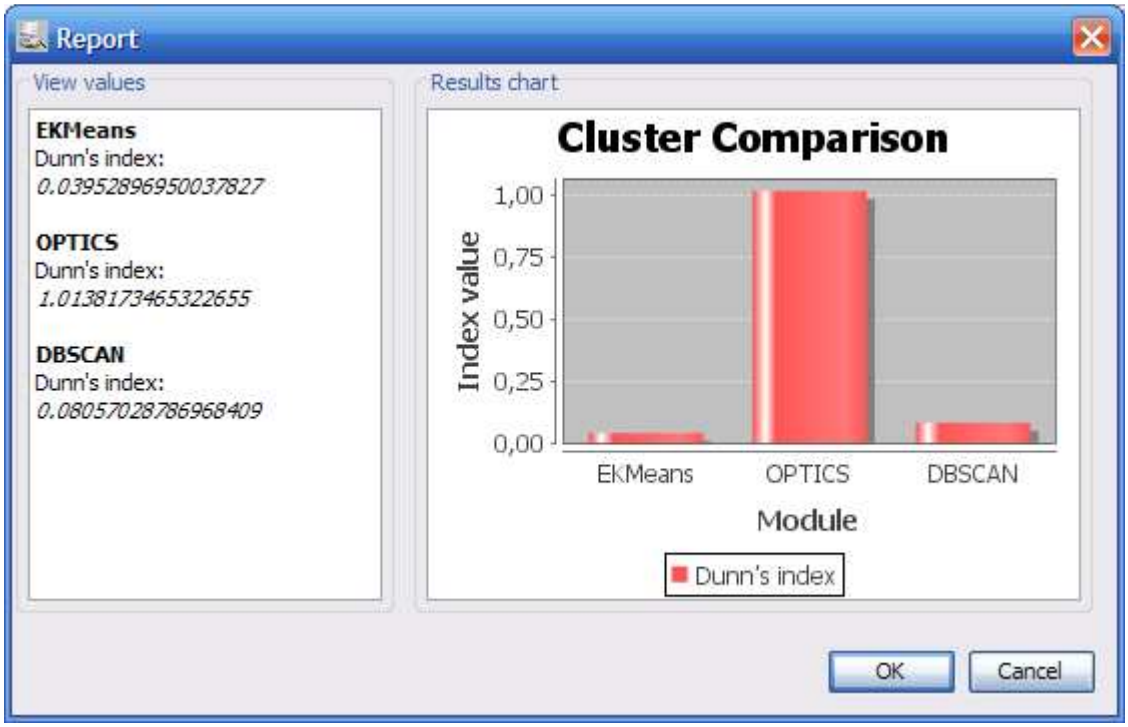
Nyní nastavíme shlukování. Nastavení modulu otevřeme stejným způsobem, jako tomu bylo u *Select Data*. U prvního vybereme algoritmus *Enhanced K-Means*, jako primární klíč vybereme atribut *ID*, *Number of Clusters* nastavíme na hodnotu 5, tato hodnota značí počet nalezených listových shluků, jedná se tedy o parametr často označovaný, jako *K*. Volbu potvrdíme a modul přejmenujeme kliknutím pravým tlačítkem a výběrem *Rename*, do dialogového okna zadáme *EKMeans* a potvrdíme. Dalšímu modulu vybereme algoritmus *DBSCAN*, vybereme primární klíč a nastavíme epsilon na

hodnotu 0,1, volbu potvrdíme a přejmenujeme na *DBSCAN*. Stejná nastavení zvolíme i pro poslední modul, ovšem tentokrát vybereme algoritmus *OPTICS*. Nyní bychom měli na ploše vidět diagram, podobný tomu, který ukazuje Obrázek 9.1.

Následuje připojení porovnávajícího modulu. Na plochu z palety přetáhneme *Cluster Comparison* a zapojíme do něj některý ze shlukovacích modulů. Na tom, který z nich to bude, nezáleží. V nastavení vybereme zbývající dva a jako metodu porovnání vybereme Dunnův Index.

Abychom mohli zobrazit jakékoliv výsledky, musíme vložit na plochu komponentu *Report*. V tomto případě report připojíme na porovnávací modul.

Celý proces nyní spustíme příkazem *Run* z kontextového menu porovnávacího modulu. Všechny předešlé komponenty dolovacího procesu budou spuštěny automaticky. Jakmile celý proces proběhne, otevřeme report. Obrázek 9.2 ukazuje, co bychom nyní měli vidět.

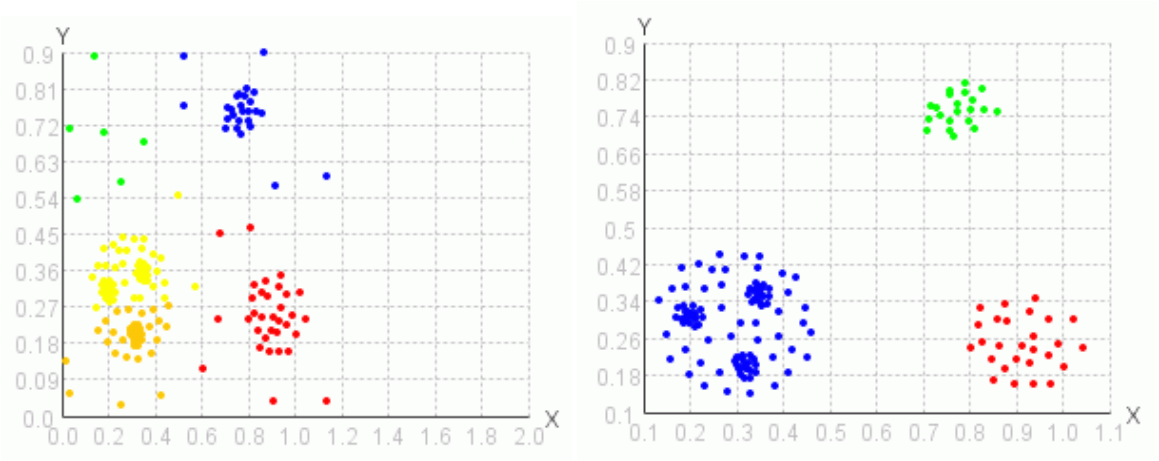


Obrázek 9.2: Porovnání výsledků shlukování

Na levé straně vidíme vypočítané výsledky, vpravo zase graf, s jehož pomocí si můžeme tyto výsledky lehce porovnat. V tomto případě to vypadá, že na plné čáře zvítězil algoritmus OPTICS. Protože jsme použili pouze dvojrozměrná data, můžeme si ověřit kvalitu tohoto porovnání vizuálně, tak že si výsledky algoritmů prohlédneme a porovnáme si výsledek OPTICS a Enhanced K-Means.

Systém nedovoluje vložení více prvků *report* na pracovní plochu, proto musíme k prohlížení jednotlivých výsledků *report* přepojovat mezi jednotlivými komponentami. První, co uděláme, je, že smažeme linku mezi porovnávacím modulem a reportem. Report nyní připojíme na *EKMeans*, otevřeme jej a přepneme na kartu *Numerical Attributes*. V grafu vidíme, že algoritmus nalezl 5 zadaných shluků (Obrázek 9.3a). V případě shluku označeného zelenou barvou se jedná o velice

nekompaktní shluk, vzdálenosti mezi shluky jsou nízké, do shluků jsou zařazeny i body z řídkých oblastí.



Obrázek 9.3: a) výsledek shlukování algoritmu Enhanced K-Means, b) Výsledek shlukování algoritmu OPTICS

Pokud se stejným způsobem podíváme na výsledek algoritmu OPTICS (Obrázek 9.3b), vidíme, že 3 nalezené shluky jsou dobře odděleny, vzdálenosti mezi shluky jsou vysoké, shluky vznikly pouze v hustých oblastech.

Definice shluku nám říká, že objekty uvnitř shluku jsou si podobné (blízké) a objekty z různých shluků si nejsou podobné (jsou si vzdálené). U našeho příkladu tuto definici rozhodně lépe naplňuje výsledek algoritmu OPTICS. Na první pohled lze vidět, že nalezl kompaktnější shluky, mezi kterými jsou větší vzdálenosti. Z tohoto důvodu je u něj vyšší i vypočítaná hodnota Dunnova Indexu.

10 Závěr

10.1 Zhodnocení

Cílem této diplomové práce bylo nastudovat problematiku získávání znalostí z databází, seznámit se s metodami shlukové analýzy, metodami prezentace jejích výsledků a systémem pro dolování z dat vyvíjeným na FIT VUT v Brně. Dále pak implementovat vybrané metody shlukové analýzy a začlenit je v podobě modulu do stávajícího systému. Jedná se o systém vyvíjený studenty v rámci diplomových a bakalářských prací, proto bylo nutné před návrhem a implementací nového modulu nastudovat práce studentů, kteří se na vývoji tohoto systému v minulých letech podíleli [8] [9] [7] [15].

Bylo také potřeba nastudovat jazyk DMSL, který je použit ke specifikaci celého dolovacího procesu a slouží také jako prostředek pro uložení získaných znalostí. Moduly využívají především dvě jeho součásti a to element *DataMiningTask* k uložení parametrů dolovací úlohy a element *Knowledge* k uložení znalosti. Mezi další použité technologie patří Oracle Data Mining (ODM), především jeho podpora pro shlukovou analýzu, a také s ním související rozhraní JDM (JSR-73).

Výsledkem práce je modul shlukové analýzy, jehož vývoj probíhal z části ve spolupráci s Bc. Martinem Hlostou [16]. Podstatná část však byla řešena samostatně a to implementace algoritmů založených na ODM, vizualizace výsledku celého modelu, vizualizace založená na nominálních a na intervalových proměnných. Jako rozšíření vznikl modul validace výsledků, který implementuje porovnání založené na relativních kritériích.

Při návrhu obou těchto modulů byl kladen důraz a udržovatelnost a rozšiřitelnost. Návrh probíhal s ohledem na základní principy objektově orientovaného návrhu, přičemž bylo využito množství návrhových vzorů. Modul shlukové analýzy je snadno rozšiřitelný o nové algoritmy a vzdálenostní funkce, modul pro validaci výsledků zase o další validační indexy.

Shlukování je v systému snadno použitelné a nabízí výběr z 5 dostupných algoritmů, z nichž 3 jsou nezávislé na použitém databázovém systému, 2 jsou závislé na ODM (Enhanced K-Means a O-Cluster). Modul nabízí bohaté možnosti vizualizace výsledného modelu. Kromě obecné vizualizace může každý algoritmus vizualizaci rozšířit o další možnosti, které jsou pro něj specifické.

10.2 Další rozvoj projektu

Rozšíření modulu shlukové analýzy

Modul shlukové analýzy již teď obsahuje výběr z pěti algoritmů. Poskytuje ale také dobrý základ pro připojení dalších algoritmů k systému a to včetně vlastní vizualizace.

Změna práce s grafem komponent

Při současné implementaci systému je grafem komponent vždy strom. Kvůli tomuto omezení vznikly problémy při implementaci modulu porovnání pro shlukovou analýzu. Systému by se však také hodil například modul pro sumarizaci výsledků klasifikace. Z tohoto důvodu by bylo vhodné, pokud by komponenty na ploše mohly vytvářet acyklický graf. Dále by bylo vhodné rozšířit *Report* tak, aby mohl být na plochu umístěn vícekrát.

Zjednodušení předzpracování dat

Práce s komponentami *VIMEO* a *Transformations* je příliš složitá. Uživatel, který systém dobře nezná, s těmito komponentami nebude schopen pracovat.

Odladění aplikace a odstranění chyb

Příliš mnoho chyb systému je dle mého názoru nejpodstatnější věc, která znemožňuje jeho reálné nasazení. Některé chyby je v systému navíc obtížné vystopovat. Např. metoda *run()* třídy *MiningPiece* má podle svého popisu vracet hodnotu *false* nejen v případě, kdy její běh skončí chybou, ale i tehdy, kdy již má k dispozici výsledky z předchozího běhu. Chyba se tedy může projevit až v následujícím modulu.

Dalším častým zdrojem chyb v systému je kopírování instancí modulů (návrhový vzor prototype), při kterém moduly v systému většinou bohužel vytvářejí tzv. shallow copies. Tak je tomu proto, že ve třídě *MiningPiece* je sice metoda *clone()* implementována, ale pouze tak že volá stejnojmennou metodu nadřazené třídy. Pokud pak např. umístíme na plochu 2 instance komponenty *Select Data*, je druhá instance kopií té první. Obě instance však sdílejí reference na své privátní proměnné. Např. kolekce *destConnections* pak bude pro obě instance totožná a bude obsahovat moduly připojené k oběma instancím *Select Data*, to pak může vést na nečekané chování. Otázka zde tedy zní, zdali chceme opravdu kopírovat již vytvořené instance, nebo při přetažení modulu na plochu vytvořit instanci novou.

Internacionalizace aplikace

Popisky uživatelského rozhraní a chybové hlášky jsou v současné verzi systému psány pouze v angličtině. Některé jsou vypisovány přímo ze zdrojového kódu, což není obecně dobrý přístup. Pokud bychom nalezli gramatickou chybu, či překlep, je nutné v kódu tyto textové řetězce složitě dohledávat. Platforma NetBeans nabízí možnost umisťování textových řetězců do speciálních souborů, tzv. bundles. Pro každý balíček jsou pak všechny textové řetězce uloženy na jednom místě. To pak poskytuje i další výhodu v podobě možnosti překladu těchto souborů např. do češtiny. Jazyk rozhraní si pak v takovém případě může uživatel zvolit sám, nebo je vybrán automaticky v závislosti na nastavení jazykového prostředí v operačním systému uživatele.

Literatura

[1] **Zendulka, J., a další.** *Získávání znalostí z databází - studijní opora.* Brno : FIT VUT v Brně, 2009.

[2] **Han, J. a Kamber, M.** *Data Mining: Concepts and Techniques. Second Edition.* San Francisco : Elsevier Inc., 2006. ISBN: 978-1-55860-901-3.

[3] **Xu, R. a Wunsch, D.** *Clustering.* Hoboken : John Wiley & Sons, Inc., 2009. ISBN: 978-0-470-27680-8.

[4] *Hierarchical K-means: an algorithm for centroids initialization for K-means.* **Arai, K. a Barakbah, A.** Saga, Japonsko : Saga University, 2007. ISSN: 03856186.

[5] **Oracle Corporation.** K-Means. *Data Mining Concepts.* [Online] 2009. [Citace: 29. Prosinec 2009.] http://download.oracle.com/docs/cd/B28359_01/datamine.111/b28129/algo_kmeans.htm.

[6] *O-Cluster: Scalable Clustering of Large High Dimensional Data Sets.* **Milenova, B. a Campos, M.** Burlington : Oracle, 2002.

[7] **Šebek, M.** Rozšíření funkcionality systému pro dolování z dat na platformě NetBeans. *Diplomová práce.* Brno : FIT VUT v Brně, 2009.

[8] **Krásný, M.** Systém pro dolování z dat v prostředí Oracle. *Diplomová práce.* Brno : FIT VUT v Brně, 2008.

[9] **Máder, P.** Dolovací moduly systému pro dolování z dat v prostředí Oracle. *Diplomová práce.* Brno : FIT VUT v Brně, 2009.

[10] **Kotásek, P.** DMSL: The Data Mining Specification Language. *Disertační práce.* Brno : FIT VUT v Brně, 2003.

[11] **Hlosta, M.** Modul pro shlukovou analýzu systému pro dolování z dat. *Diplomová práce.* Brno : FIT VUT v Brně, 2010.

[12] **Asuncion, A. a Newman, D.J.** *UCI Machine Learning Repository.* [Online] University of California, School of Information and Computer Science, 2007. [Citace: 3. 4 2010.] http://www.ics.uci.edu/~mlearn/MLRepository.html.

[13] **Object Refinery Ltd.** JFreeChart. [Online] Object Refinery Ltd., 2009. [Citace: 05. 05 2010.] http://www.jfree.org/jfreechart/.

[14] **Richet, Yann.** JMathPlot. [Online] 2009. [Citace: 04. 05 2009.] http://code.google.com/p/jmathplot/.

[15] **Rapid-I.** Rapid Miner. *Rapid-I.* [Online] Rapid-I, 2010. [Citace: 15. 5 2010.] http://rapidminer.com/.

[16] **Henkl, T.** Dolovací moduly systému pro dolování z dat na platformě NetBeans. *Diplomová práce.* Brno : FIT VUT v Brně, 2009.

[17] **NetBeans.** NetBeans Platform. [Online] [Citace: 1. Leden 2010.] http://platform.netbeans.org/.

[18] **Gamma, E., a další.** *Design Patterns : Elements of Reusable Object-Oriented Software.* : Addison-Wesley Professional, 1998. ISBN 978-0201633610.

Seznam příloh

Příloha A: Obsah přiloženého CD

- *Src* – zdrojový kód
- *Data Miner* – spustitelná aplikace
- *Doc/clusteringmodule* – dokumentace modulu shlukové analýzy
- *Doc/clustercomparison* – dokumentace porovnávacího modulu
- *Sample* – soubory k ukázkovému příkladu
- *Readme.txt* – soubor s obsahem CD a dodatečnými informacemi